Data storage and preservation

Sébastien Ponce
sebastien.ponce@cern.ch
CERN

Thematic CERN School of Computing 2019

Outline

Storage devices

A variety of storage devices

Main differences

- Capacities from 1 GB to 10 TB per unit
- Prices from 1 to 300 for the same capacity
- Very different reliability
- Very different speeds

A variety of storage devices

Main differences

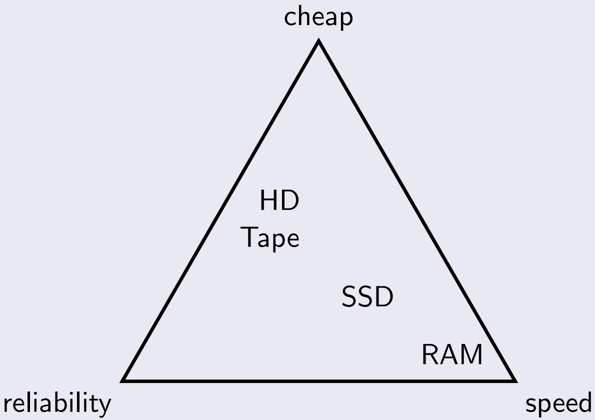
- Capacities from 1 GB to 10 TB per unit
- Prices from 1 to 300 for the same capacity
- Very different reliability
- Very different speeds

Typical numbers in 2019

	Capacity per unit	Latency	$/TB	Speed	reliability
RAM	16 GB	10 ns	7000 $	$10\,\mathrm{GB\,s^{-1}}$	volatile
SSD	500 GB	10 µs	200 $	$1\,\mathrm{GB\,s^{-1}}$	poor
HD	6 TB	3 ms	25 $	$150\,\mathrm{MB\,s^{-1}}$	average
Tape	20 TB	100 s	20 $	$500\,\mathrm{MB\,s^{-1}}$	good

A variety of storage devices

You cannot have everything



Reliability in real world (CERN)

For disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in 10^{14} bits read/written
  - for 10GB files, that's one file corrupted per 1000 files written

Reliability in real world (CERN)

For disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in 10^{14} bits read/written
  - for 10GB files, that's one file corrupted per 1000 files written

For tapes

- probability of losing a tape per year : 10^{-4}
  - and you recover most of the data on it
  - net result is 10^{-7} file loss per year
- one unrecoverable bit error in 10^{19} bits read/written
  - for 10GB files, that's one file corrupted per 100M files written

CERN School of Computing

Parallelizing files' storage

Why to parallelize storage ?

to work around limitations

- individual device speed (think disk)
  - a file is typically stored on a single device
- network cards' speed
  - 1 Gbit network still present
  - network congestion on a node reduces bandwidth per stream
- core network throughput
  - switches / routers are expensive
  - machines may have less throughput than their card(s) allow(s)
- hot data congestions
  - and the black hole it can generate
  - as slower tranfers allow to accumulate more transfers

Parallelizing through striping

Main idea

- use several devices in parallel for a single stream
- moving the limitations up by summing performances

Basic striping : Divide and conquer for storage

- split data into chunks aka stripes on different devices
- access in parallel

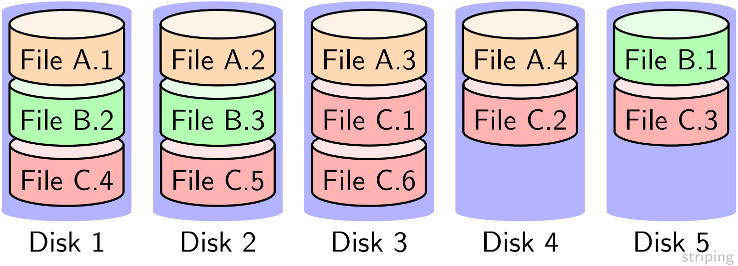
Parallelizing through striping

Main idea

- use several devices in parallel for a single stream
- moving the limitations up by summing performances

Basic striping : Divide and conquer for storage

- split data into chunks aka stripes on different devices
- access in parallel



RAID 0

RAID

- stands to "Redundant Array of Inexpensive Disks"
- set of configurations that employ the techniques of striping, mirroring, or parity to create large reliable data stores from multiple general-purpose computer hard disk drives (Wikipedia)

Useful RAID levels

- RAID 0 striping
- RAID 1 mirroring
- RAID 5 parity
- RAID 6 double parity

Can be implemented in hardware or software

RAID 0

RAID

- stands to “Redundant Array of Inexpensive Disks”
- set of configurations that employ the techniques of striping, mirroring, or parity to create large reliable data stores from multiple general-purpose computer hard disk drives (Wikipedia)

Useful RAID levels

RAID 0	striping	
RAID 1	mirroring	See Data Safety part
RAID 5	parity	See Data Safety part
RAID 6	double parity	See Data Safety part

Can be implemented in hardware or software

RAID versus RAIN

RAIN

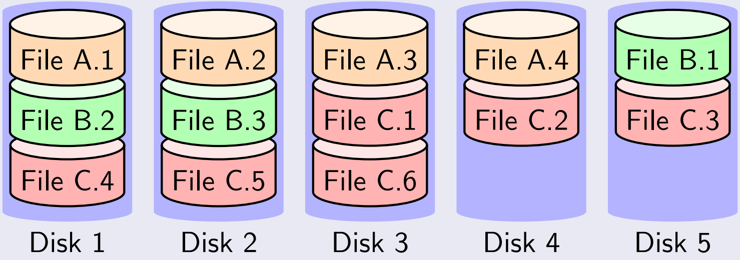
- Redundant Array of Inexpensive Nodes
- similar to RAID but across nodes

Main interest

- tackle also the network limitations
- when used for redundancy, improves reliability
- more on this in subsequent lecture

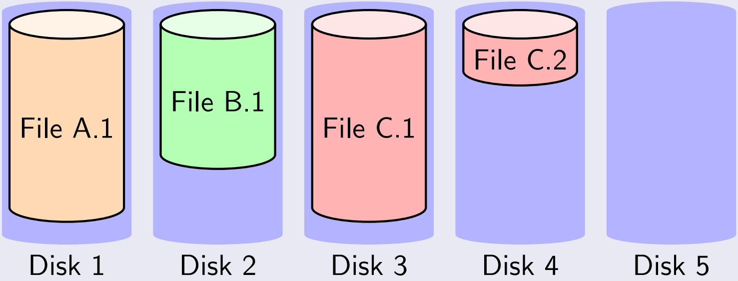
Practical striping - the stripe size

Desired picture



Practical striping - the stripe size

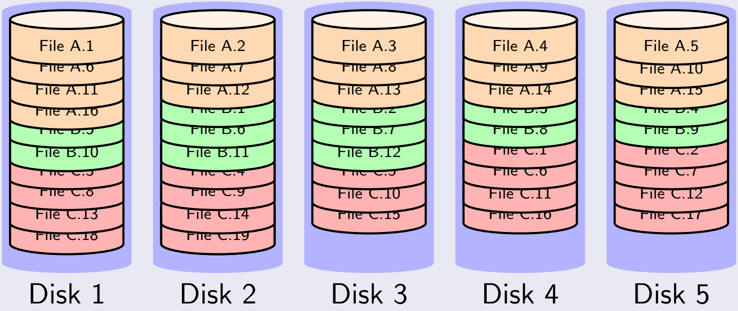
Stripes too big



RAID has practically not effect

Practical striping - the stripe size

Striped too small



RAID will only kill performance by forcing disk to seek far too often

How to choose the stripe size

size of the stripe

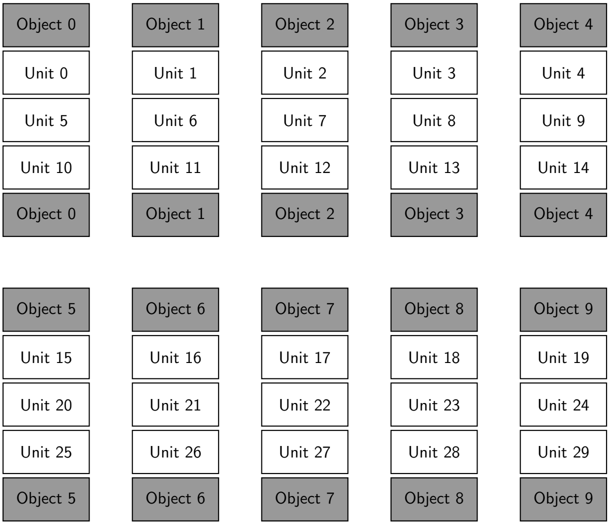
- must be as small as possible to let small reads benefit from parallelization
- must not be too small
  - to avoid having to deal with too much metadata
  - to avoid too much disk seeking

A generic solution for the stripe size

Idea

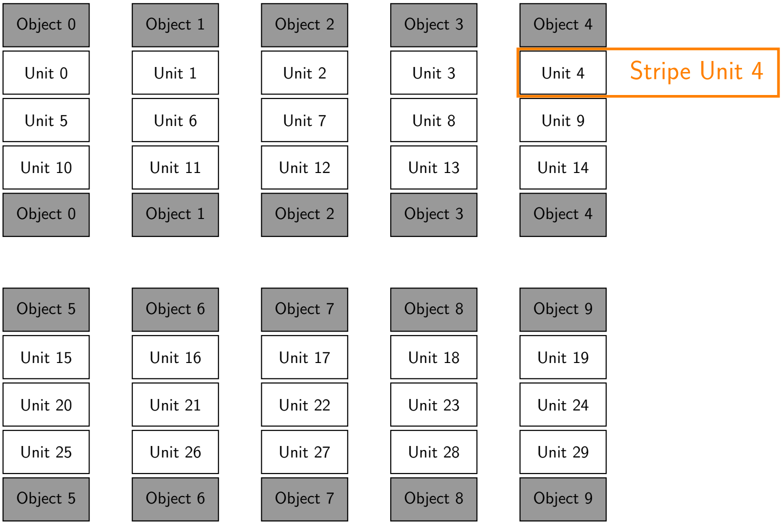
- disentangle "stripe size" from "object size"
- "stripe size" is the size of one slice of data
- "object size" is the size of one block of data on disk
- several stripes are put together into one bigger object

Ceph striping

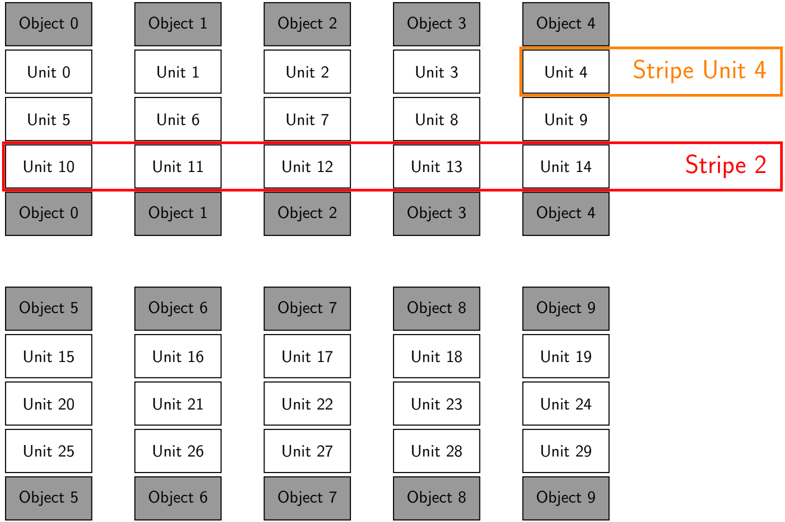


thematic CSC CERN School of Computing

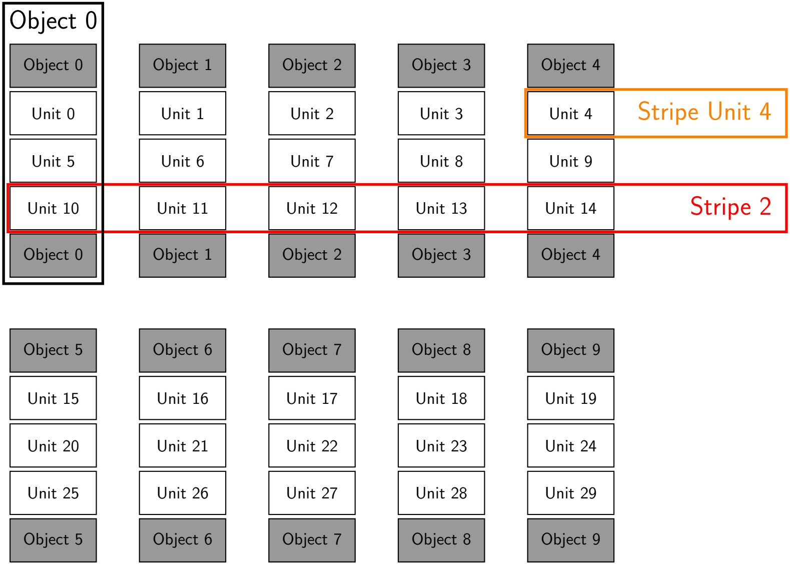
Ceph striping



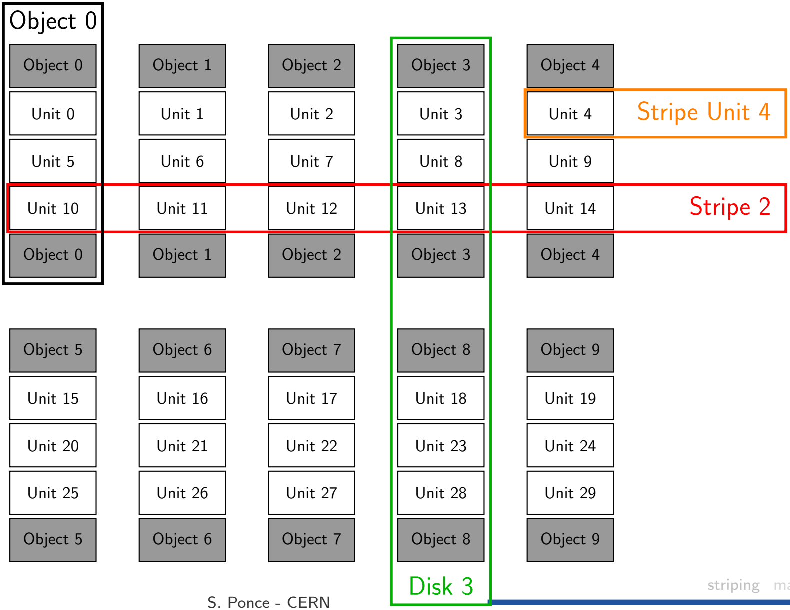
Ceph striping



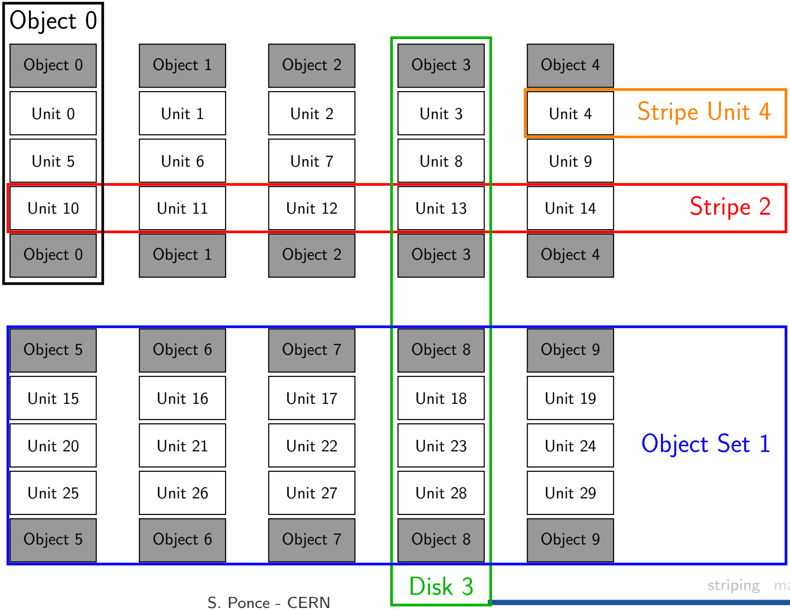
Ceph striping



Ceph striping



Ceph striping



Practical striping - number of disks

Why to have many

- to increase parallelism
- to get better performances

Why to have few

- to limit the risk of losing files
- as losing a disk now means losing all files of all disks
- if p is the probability to lose a disk
  the probability to lose one in n is $p_n = np(1-p)^{n-1} \sim np$

A generic solution for the number of disks

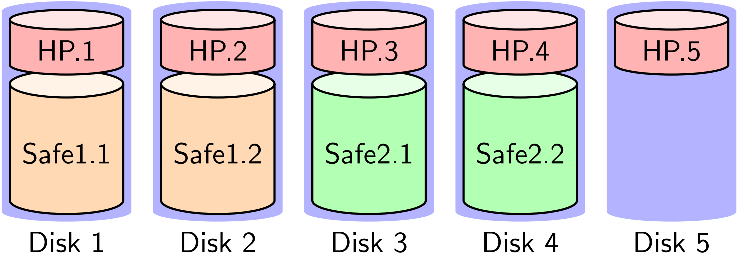
Idea

- disentangle "nb disks" from "nb stripes"
- do not use all disks for all files
- adapt your number of disks to each file
  - more disks for high performance files
  - less disks for more safety

A generic solution for the number of disks

Idea

- disentangle “nb disks” from “nb stripes”
- do not use all disks for all files
- adapt your number of disks to each file
  - more disks for high performance files
  - less disks for more safety



Going further : Map/Reduce

What do we have with striping ?

- striping allows to distribute server I/O on several devices
- but client still faces the total I/O
- and CPU is not distributed

Map/Reduce Idea

- send computation to the data nodes
- "the most efficient network I/O is the one you don't do"

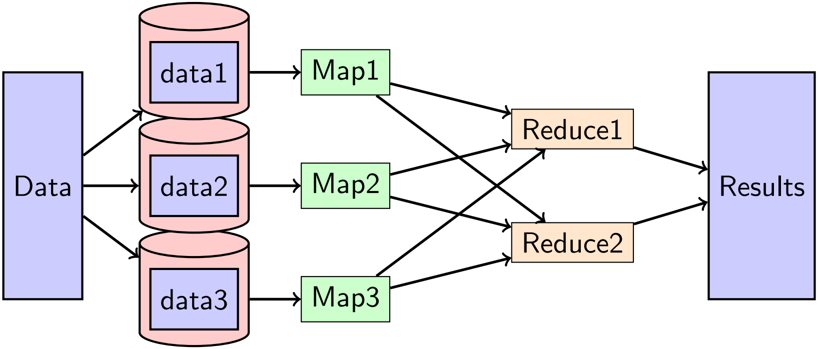
Map/Reduce Introduction

Schema

Map processes data locally and returns key/value pairs

Shuffle sends output to *Reduce* step based on output key

Reduce merges partial results to final output

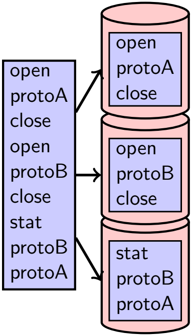


Example : usage of protocols from logs

open
protoA
close
open
protoB
close
stat
protoB
protoA

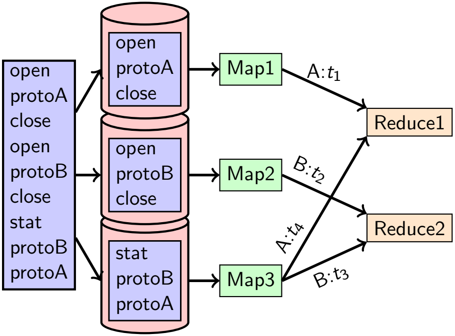
Example : usage of protocols from logs

1. split logs on different nodes



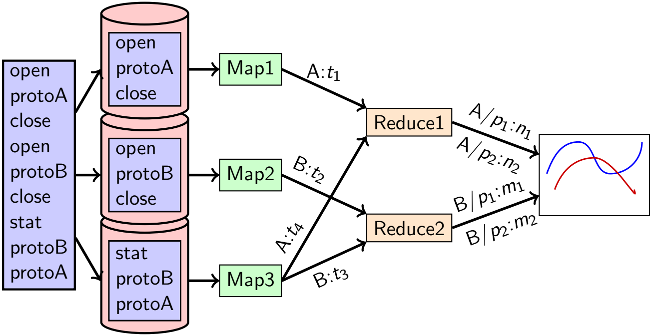
Example : usage of protocols from logs

1. split logs on different nodes
2. parse and extract key/value (*Protocol*/*Date*) in Map



Example : usage of protocols from logs

1. split logs on different nodes
2. parse and extract key/value (*Protocol*/*Date*) in Map
3. count accesses and output *Date*/*nb accesses* in Reduce
4. build graph

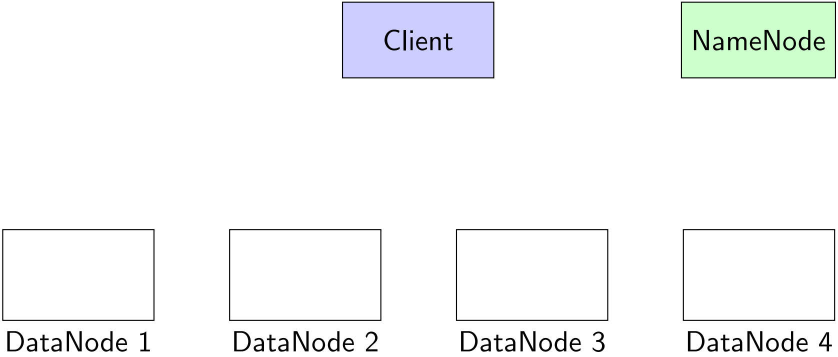


The case of Hadoop, HDFS

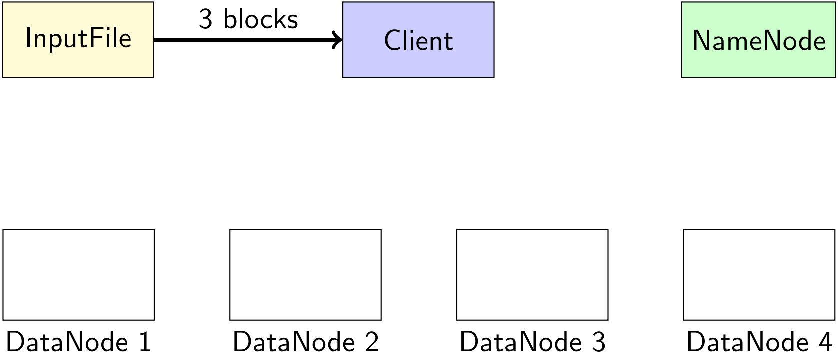
HDFS : the Hadoop Distributed FileSystem

- Distributed
  - files split in blocks spread over the cluster
  - blocks are typically 128MB
- Redundant
  - mirroring, 3 copies by default
  - erasure coding from version 3.0 on
- Hardware aware

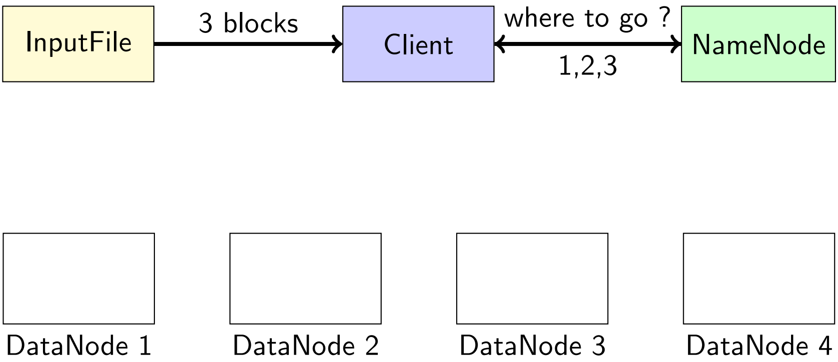
Parallel Read/Write in HDFS



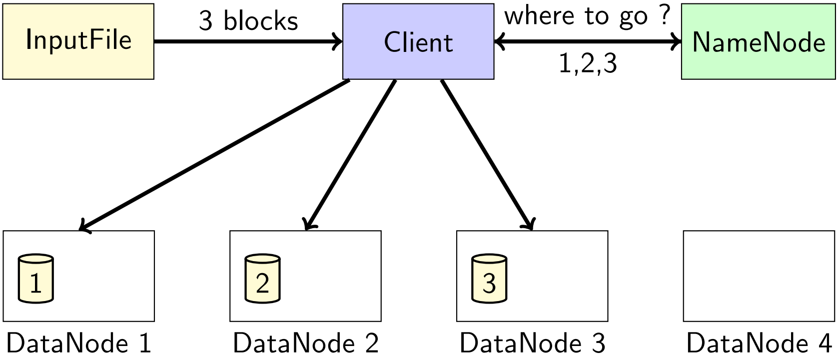
Parallel Read/Write in HDFS



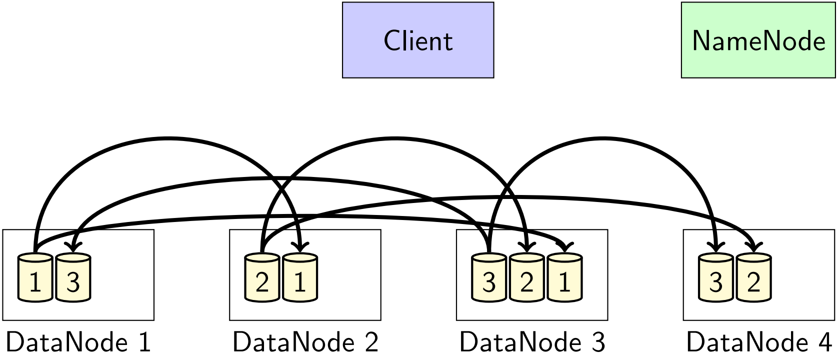
Parallel Read/Write in HDFS



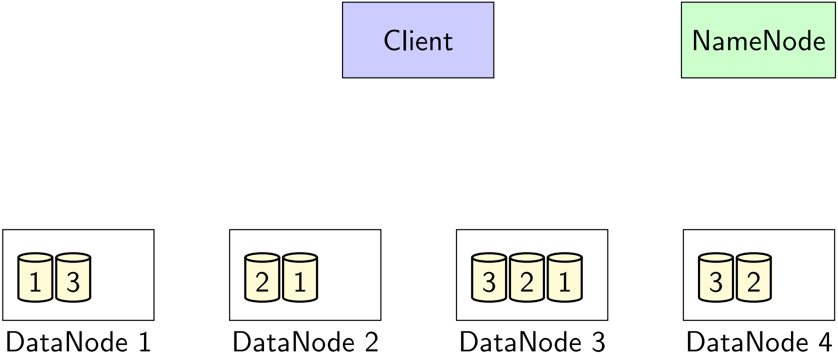
Parallel Read/Write in HDFS



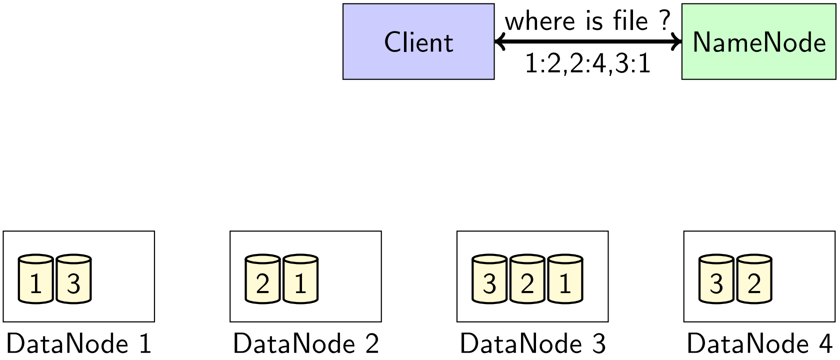
Parallel Read/Write in HDFS



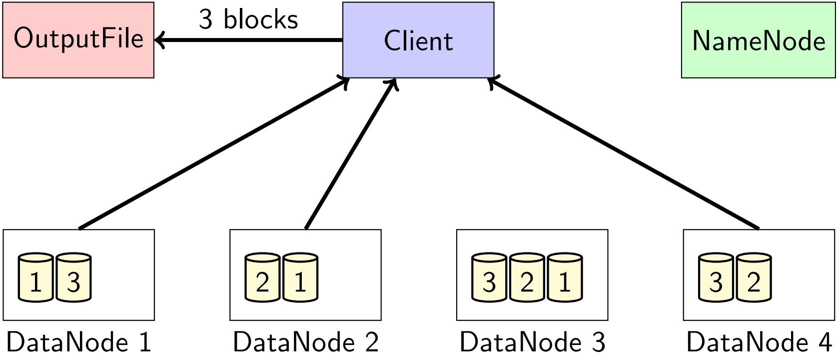
Parallel Read/Write in HDFS



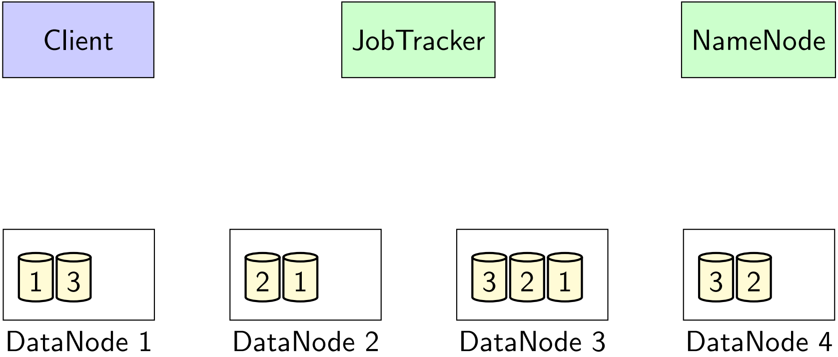
Parallel Read/Write in HDFS



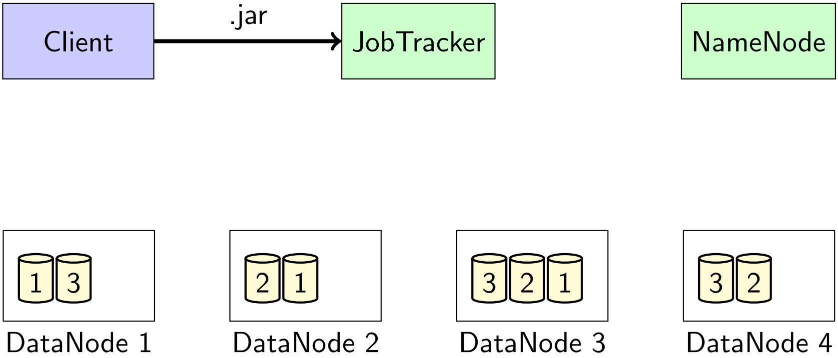
Parallel Read/Write in HDFS



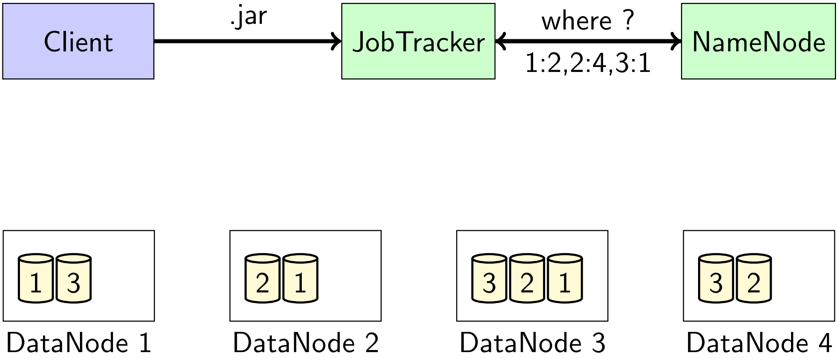
Parallel Map/Reduce



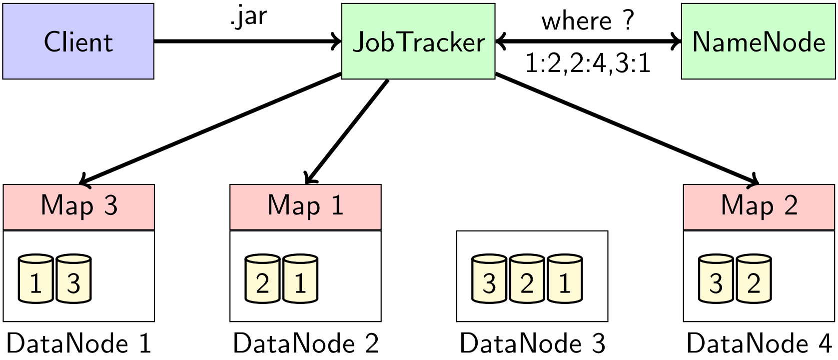
Parallel Map/Reduce



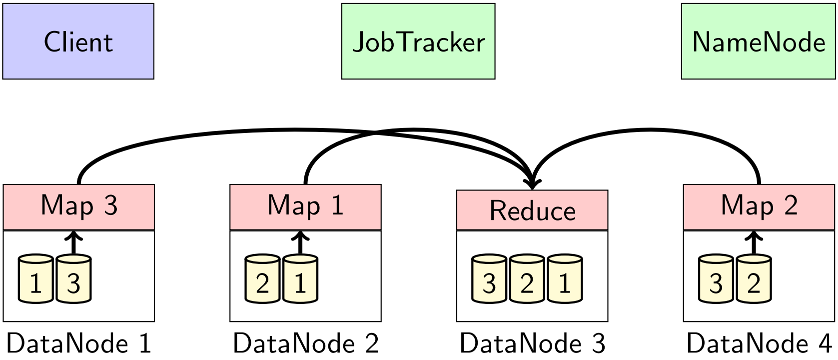
Parallel Map/Reduce



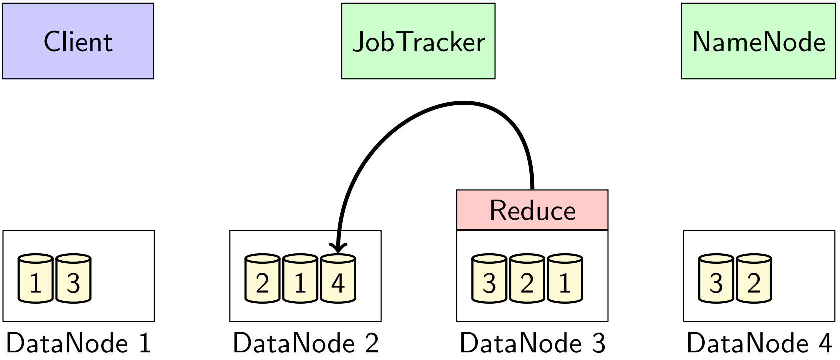
Parallel Map/Reduce



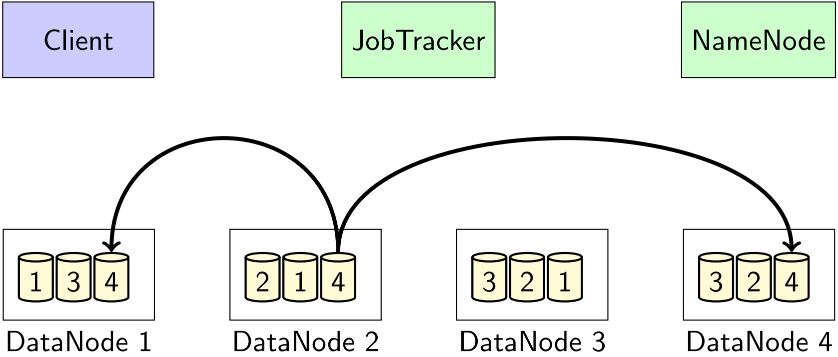
Parallel Map/Reduce



Parallel Map/Reduce



Parallel Map/Reduce



Risks of data loss and corruption

Risks for my data - Hardware

For disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in 10^{14} bits read/written
  - for 10GB files, that's one file corrupted per 1000 files written

For tapes

- probability of losing a tape per year : 10^{-4}
  - and you recover most of the data on it
  - net result is 10^{-7} file loss per year
- one unrecoverable bit error in 10^{19} bits read/written
  - for 10GB files, that's one file corrupted per 100M files written

Risks for my data - Software

BUGS !

- in your software
  - e.g. scheduling twice a transfer, not receiving data on the second run and overwriting the correct file with an empty one
- in your dependencies
  - e.g. the transfer protocol used does not support checksum and data may be corrupted by TCP (checksum is only 16 bit, one corrupted packet in 65536 will go through)
- in the OS or common libraries
  - e.g. libc locks not being atomic
- in the hardware - that is in the micro code running inside
  - e.g. RAID controllers
- in your admin tools
  - e.g. recycling a tape that was not empty

Risks for my data - Human factor

Real life cases that went wrong

- reinstall (and wipe) old machine p23425a4752
  - Oh no, I actually meant p42532a8779... bad cut and paste
- rm -rf /top/data/alltimes /2015/04/crap
  - one space too much and all data are gone....
- activate garbage collection on pool XYZ, it's full
  - wasn't it tape backed up ? no ? oups....

Risks for my data - conclusion

Risks for my data - conclusion

You will lose/corrupt data !

Risks for my data - conclusion

You will lose/corrupt data !

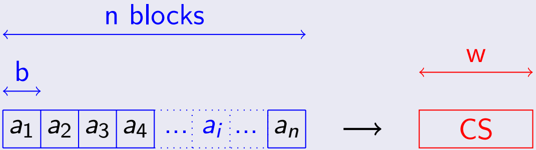
- better to be able to know when and what
- even better if you can repair

Data consistency

Checksum

Definition

“small-size datum from a block of digital data for the purpose of detecting errors“



Most basic checksum : data size

Computation

$$\boxed{a_1}\boxed{a_2}\boxed{a_3}\boxed{a_4}\ \dots\ a_i\ \dots\ \boxed{a_n} \longrightarrow \boxed{n}$$

$$b = 8\,\text{bit} \quad w = 64\,\text{bit} \quad CS = n$$

Pros and Contra

- easy to compute
- detects erasures and additions
- does not detect any corruption

thematic CERN School of Computing

Basic checksum : sum/xor

Computation

a_1	a_2	a_3	a_4	$\dots$	a_i	$\dots$	a_n	$\longrightarrow$	$\sum a_i$

$$b \quad w = b \quad CS = \sum_{i=1}^{n} a_i$$

Pros and Contra

- easy to compute
- detects most corruptions
- does not detect any inversions/change of order

Adler like checksums

Computation

$$a_1 \; a_2 \; a_3 \; a_4 \; \ldots \; a_i \; \ldots \; a_n \longrightarrow \sum a_i \; \sum i a_i$$

$$b = 8\,\text{bit} \quad w = 32\,\text{bit} \quad CS_{high} = \sum_{i=1}^{n} a_i \quad CS_{low} = \sum_{i=1}^{n} i a_i$$

Pros and Contra

- easy to compute
- detects most corruptions and inversions
- weak for small files
- easy to fake in case of intentional corruption

(Crypt)Analysis of adler

Weaknesses

- 32 bits is short
  - one per 4 billion corruption will go through
- it's actually worse for small files
  - all bits of the sum are not even used for less than 256 bytes
- they can be easily bypassed
  - one can easily change the last 16 bytes and reach any checksum
  - so intentional corruptions are not covered

Cryptographic checksums

What is it ?

- checksums that cannot be faked (easily)
- they are based on non reversible cryptographic functions

Most used ones

md5 1991, 128 bits, by Rivest. Not considered secure anymore as complete collisions have been discovered.

sha1 1995, 160 bits, by NSA. Collision in 2^{61} operations

sha256 2001, 256 bits, by NSA. Collision in 2^{128} operations

sha512 2001, 512 bits, by NSA. Collision in 2^{256} operations

Drawback

- more costful to compute
- although modern processors have dedicated instructions

Comparison of main checksums

Name	MB/s on intel core 2	Cycles Per Byte
Adler32	920	1.9
MD5	255	6.8
SHA-1	153	11.4
SHA-256	111	15.8
SHA-512	99	17.7

Practical usage of checksums

Simple approach

- compute checksum in memory when creating/writing file
- store checksum in a DB
- check it in memory on full file reads

Problems

- corrupted data only found when read back, unnoticed otherwise
- one needs to fully read the file to be able to check
- file updates not supported. Need to read back the whole file
- file append suffers the same limitation
- multi stream, out of order writing not supported
- losing the DB loses all checksums
- renaming a file implies changing the entry in the DB, with double commit issue

Recommendations

When to compute checksums

- opportunistically
- plus regular scans of the whole data set

How to store checksums

- always close to the file
- next to it on the filesystem
- in most cases, using external attributes

Data safety

How to correct corrupted data ?

So far

- checksum allow corruption detection
- but no correction is possible
- erasure of data is not covered either
- for this we need some data redundancy

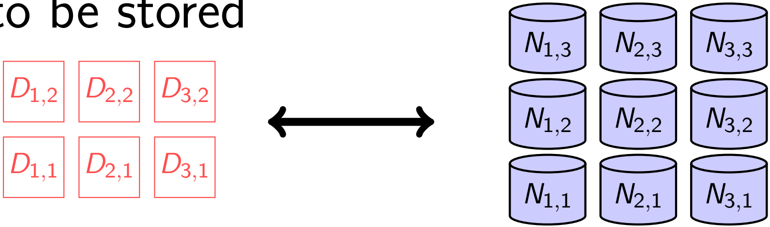
Existing techniques

- mirroring
- parity and RAID systems
- erasure coding

Context

k pieces of data to be stored

n nodes



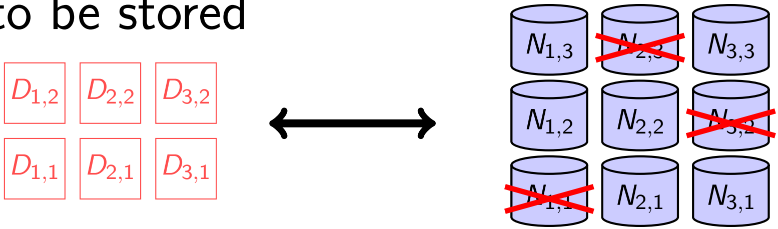
Goal

- some nodes will fail/disappear
- we want our data back anyway
- we want the cheapest price

Context

k pieces of data to be stored

n nodes



Goal

- some nodes will fail/disappear
- we want our data back anyway
- we want the cheapest price

Replication

Idea

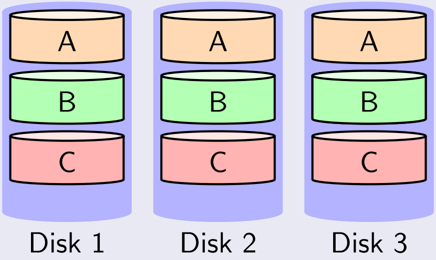
- replicate each piece of data on p disks
- you can afford losing $p-1$ disks
- you pay p times the original price

Replication

Idea

- replicate each piece of data on p disks
- you can afford losing $p-1$ disks
- you pay p times the original price

Replication - RAID 1

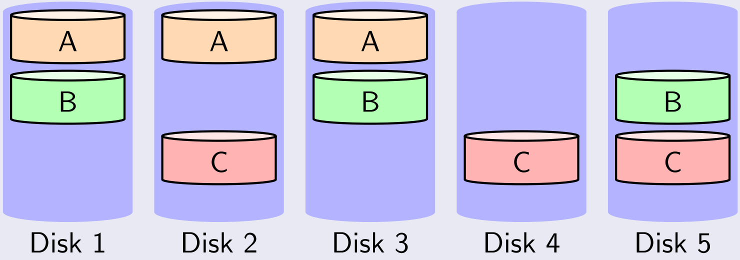


Replication

Idea

- replicate each piece of data on p disks
- you can afford losing $p - 1$ disks
- you pay p times the original price

Replication - RAID 1



Limitations of replication

With 2 replicas

- expensive : effective disk space divided by 2
- when corruption occur, no way to know which copy is corrupted
  - unless you have checksums on top

With 3 and more replicas

- horribly expensive, actually unaffordable in general

Parity

Idea

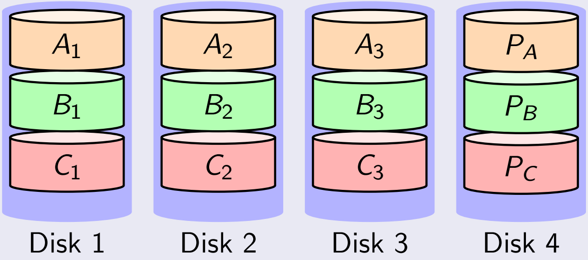
- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity

Idea

- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity Node - RAID 4

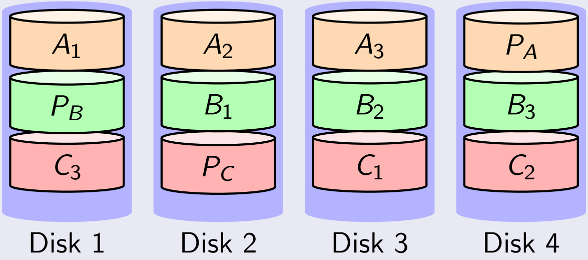


Parity

Idea

- create a parity piece of data for each k pieces (xor)
- store it as an extra piece of data
- you can afford losing 1 disk
- you pay $1 + \frac{1}{k}$ times the original price

Parity Node - RAID 5



Main issues

- writes are slightly more costful
- small updates are costful
  - one has to read back the old data to recompute parity
- only one corruption/erasure allowed
  - take care that losing a disk drastically increases the probability to lose another one
  - especially in RAID due to locality. RAIN is much better from this point of view
  - reconstructing the lost disk can take days
    e.g. 6 TB at 100 MB s^{-1} is 17 hours

Double parity

Idea : same spirit

- compute 2 parity pieces of data for each k pieces

$$P = \sum D_i \qquad Q = \sum g^i D_i$$

- you can lose 2 disks and still recover (less easily though)
- you pay $1 + \frac{2}{k}$ times the original price

Double parity

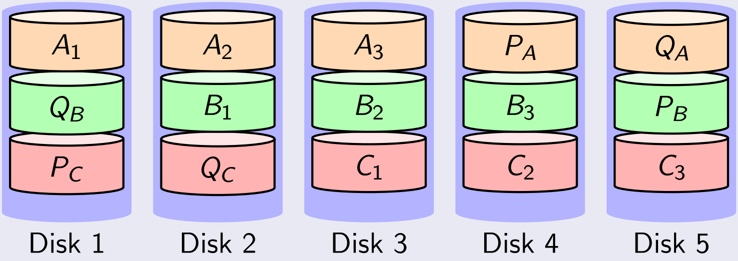
Idea : same spirit

- compute 2 parity pieces of data for each k pieces

$$P = \sum D_i \qquad Q = \sum g^i D_i$$

- you can lose 2 disks and still recover (less easily though)
- you pay $1 + \frac{2}{k}$ times the original price

Double parity - RAID 6



Cost vs Risk computations

Cost of parity

if we call r the ratio of disk space compared to data size
for k blocks with one parity, we have

$$r = 1 + \frac{1}{k}$$

Cost vs Risk computations

Cost of parity

if we call r the ratio of disk space compared to data size
for k blocks with one parity, we have

$$r = 1 + \frac{1}{k}$$

Risk reduction

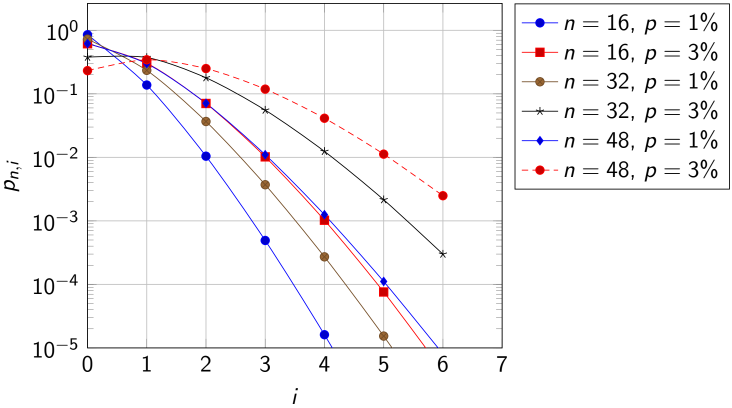
- if p the probability to lose a given disk within a given period
- we call $p_{n,i}$ the probability of losing any i disks among n

$$p_{n,0} = (1-p)^n \quad (1)$$

$$p_{n,1} = np(1-p)^{n-1} \quad (2)$$

$$p_{n,i} = \binom{n}{i} p^i (1-p)^{n-i} \quad (3)$$

Graphically, for disks within a year



Generic case (erasure coding)

Ideas

- compute more "parities" : m for each k blocks
- work on r rows of k blocks at a time

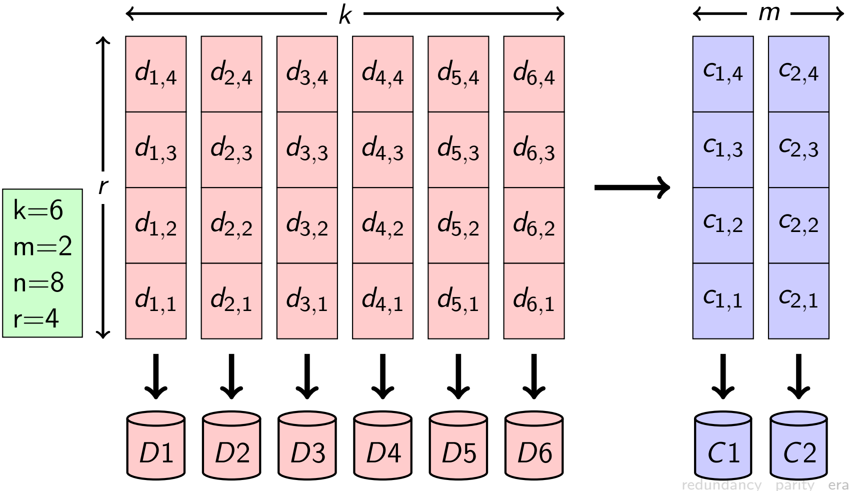
Goal

- be able to lose any m disks
- be able to reconstruct from any k disks

"Maximum Distance Separable" code (MDS)

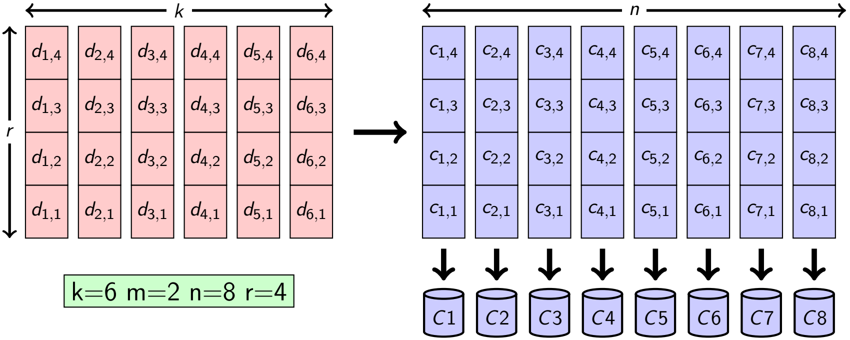
First example

Systematic code



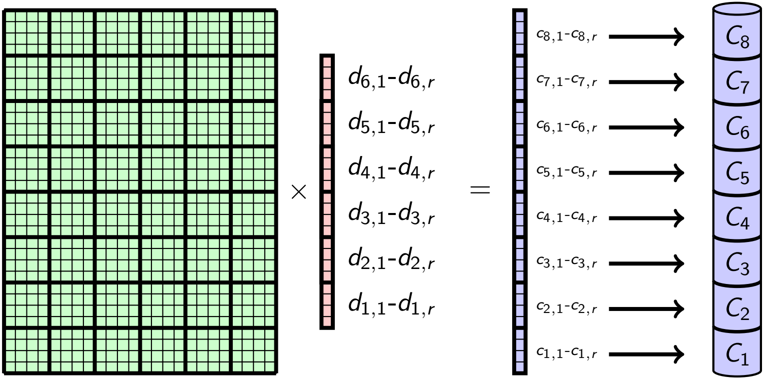
More generic example

Non systematic code



The matrix view

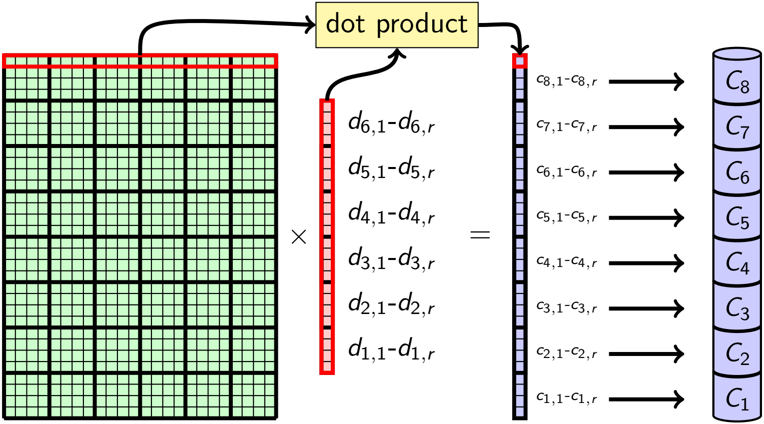
Generator Matrix G is nr x kr



CERN School of Computing

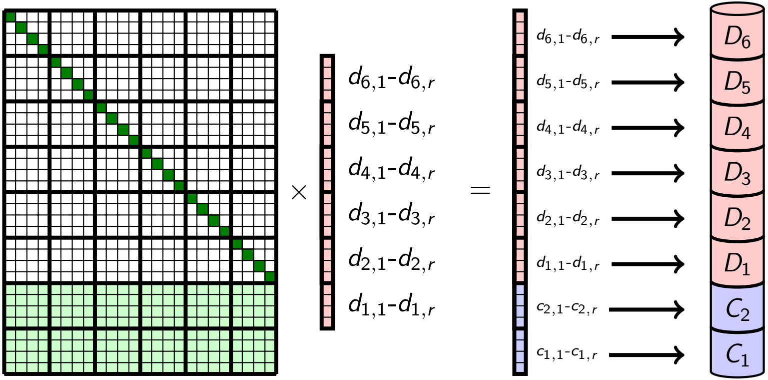
The matrix view

Generator Matrix G is nr x kr

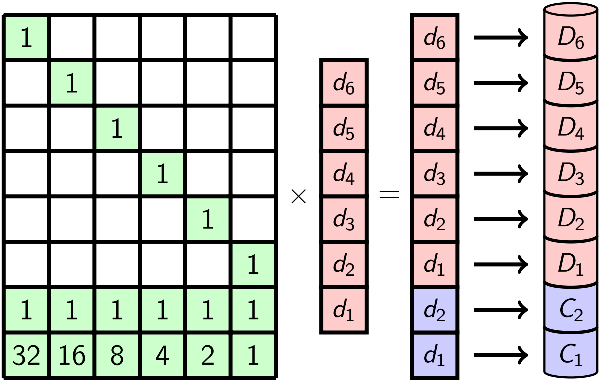


Back to systematic code

Systematic means G includes identity



Practical example : RAID 6



CERN School of Computing

Systematic Reed-Solomon Codes

1					
	1				
		1			
			1		
				1	
					1
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$

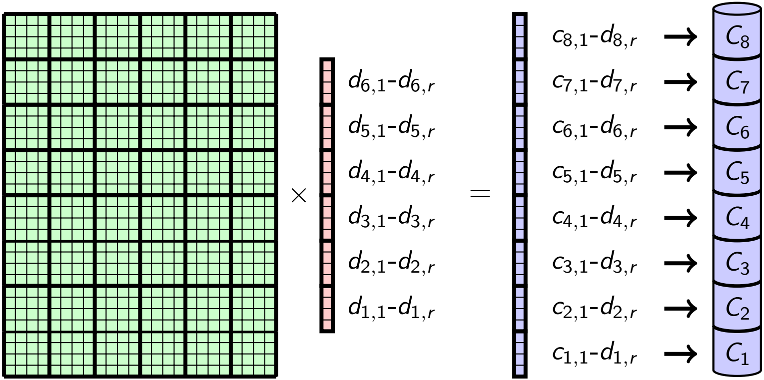
$k = 6$ and $n = 9$

- Create two sets X and Y
- X has m elements: x_0 to x_{m-1}
- Y has k elements: y_0 to y_{k-1}
- Elements in $X \cup Y$ are distinct
- $a_{i,j} = \frac{1}{x_i + y_j}$ in $GF(2^w)$ (Galois Field)

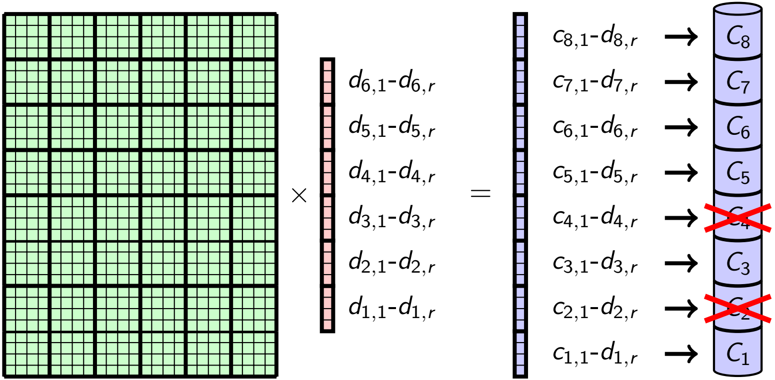
$X = 1, 2, 3$

$Y = 4, 5, 6, 7, 8, 9$

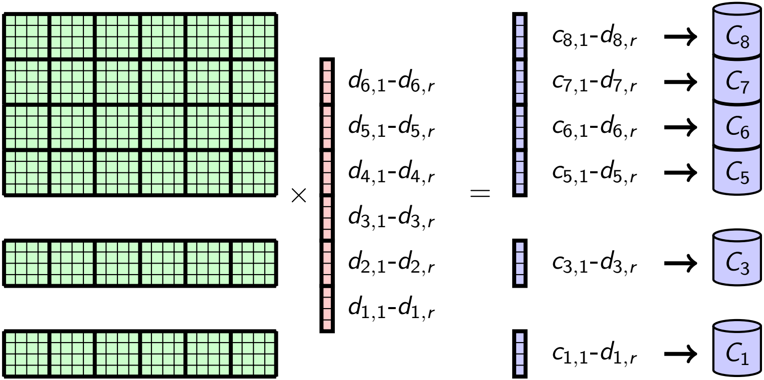
Generic reconstruction



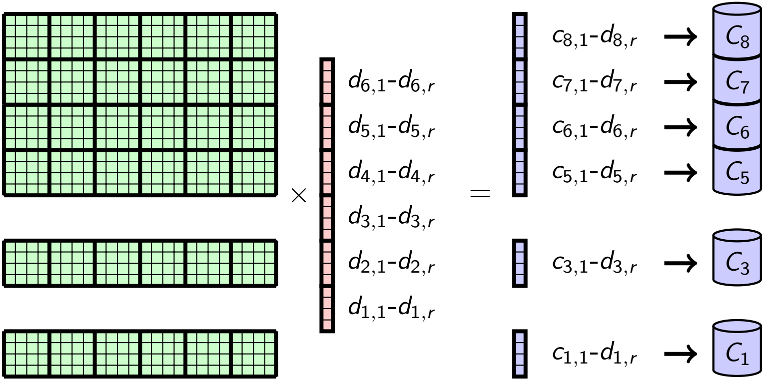
Generic reconstruction



Generic reconstruction



Generic reconstruction



$$G_{del} \times D_{del} = C_{del}$$

Generic reconstruction

$$G_{del} \times D_{del} = C_{del}$$

"Simple" matrix inversion

$$D_{del} = G_{del}^{-1} \times C_{del}$$

Reconstruction cost

Important parameters

- systematic codes are very interesting for no error case
- parity is very cheap for a single erasure/corruption : simple xor
- galois fields are more costful, especially when m grows

Reconstruction cost

Important parameters

- systematic codes are very interesting for no error case
- parity is very cheap for a single erasure/corruption : simple xor
- galois fields are more costful, especially when m grows

Improvements are possible

- pyramid codes bring improvements of reconstruction time of errors in different subparts
- Cauchy Reed-Solomon : replaces Galois fields with pure xors
- evenodd, RDP : fast single error recovery

Conclusion

Conclusion

Key messages of the day

- Take benefit of all existing devices to optimize your efficiency
- Striping solves many problems. Use it but think twice
- Map/Reduce avoids a lot of I/O when results are small
- **You will lose and corrupt data** - better be prepared
- Checksums will allow you to know
- RAID, RAIN and erasure coding will allow to recover
  - choose your security level, pay the price