Data storage and preservation

Sébastien Ponce
sebastien.ponce@cern.ch
CERN

Thematic CERN School of Computing 2019
 Outline

1. Storage devices
   - Existing devices

2. Parallelizing files' storage
   - Striping
   - Introduction to Map/Reduce

3. Risks of data loss and corruption

4. Data consistency
   - Checksums
   - Practical usage

5. Data safety
   - Redundancy
   - Parity
   - Erasure coding

6. Conclusion
Storage devices

1. Storage devices
   - Existing devices

2. Parallelizing files’ storage

3. Risks of data loss and corruption

4. Data consistency

5. Data safety

6. Conclusion
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

<table>
<thead>
<tr>
<th></th>
<th>Capacity per unit</th>
<th>Latency</th>
<th>$/TB</th>
<th>Speed</th>
<th>reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>16 GB</td>
<td>10 ns</td>
<td>7000 $</td>
<td>10 GB s⁻¹</td>
<td>volatile</td>
</tr>
<tr>
<td>SSD</td>
<td>500 GB</td>
<td>10 µs</td>
<td>200 $</td>
<td>1 GB s⁻¹</td>
<td>poor</td>
</tr>
<tr>
<td>HD</td>
<td>6 TB</td>
<td>3 ms</td>
<td>25 $</td>
<td>150 MB s⁻¹</td>
<td>average</td>
</tr>
<tr>
<td>Tape</td>
<td>20 TB</td>
<td>100 s</td>
<td>20 $</td>
<td>500 MB s⁻¹</td>
<td>good</td>
</tr>
</tbody>
</table>
A variety of storage devices

You cannot have everything

- cheap
- HD
- Tape
- SSD
- RAM

- reliability
- speed
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: \(10^{-4}\)
  - you recover most of the data on it
  - net result is \(10^{-7}\) file loss per year
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
Parallelizing files’ storage

1. Storage devices

2. Parallelizing files’ storage
   - Striping
   - Introduction to Map/Reduce

3. Risks of data loss and corruption

4. Data consistency

5. Data safety

6. Conclusion
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 transfers 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
Data storage and preservation

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
RAID 5  parity
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

File A.1
File A.2
File A.3
File A.4
File B.1
File B.2
File B.3
File C.1
File C.2
File C.3
Disk 1
Disk 2
Disk 3
Disk 4
Disk 5
Practical striping - the stripe size

Stripes too big

- File A.1 on Disk 1
- File B.1 on Disk 2
- File C.1 on Disk 3
- File C.2 on Disk 4
- Disk 5

RAID has practically not effect
**Practical striping - the stripe size**

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

<table>
<thead>
<tr>
<th>Object 0</th>
<th>Object 1</th>
<th>Object 2</th>
<th>Object 3</th>
<th>Object 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit 0</td>
<td>Unit 1</td>
<td>Unit 2</td>
<td>Unit 3</td>
<td>Unit 4</td>
</tr>
<tr>
<td>Unit 5</td>
<td>Unit 6</td>
<td>Unit 7</td>
<td>Unit 8</td>
<td>Unit 9</td>
</tr>
<tr>
<td>Unit 10</td>
<td>Unit 11</td>
<td>Unit 12</td>
<td>Unit 13</td>
<td>Unit 14</td>
</tr>
<tr>
<td>Object 0</td>
<td>Object 1</td>
<td>Object 2</td>
<td>Object 3</td>
<td>Object 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object 5</th>
<th>Object 6</th>
<th>Object 7</th>
<th>Object 8</th>
<th>Object 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit 15</td>
<td>Unit 16</td>
<td>Unit 17</td>
<td>Unit 18</td>
<td>Unit 19</td>
</tr>
<tr>
<td>Unit 20</td>
<td>Unit 21</td>
<td>Unit 22</td>
<td>Unit 23</td>
<td>Unit 24</td>
</tr>
<tr>
<td>Unit 25</td>
<td>Unit 26</td>
<td>Unit 27</td>
<td>Unit 28</td>
<td>Unit 29</td>
</tr>
<tr>
<td>Object 5</td>
<td>Object 6</td>
<td>Object 7</td>
<td>Object 8</td>
<td>Object 9</td>
</tr>
</tbody>
</table>
Ceph striping

Object 0
Unit 0
Unit 5
Unit 10
Object 0

Object 1
Unit 1
Unit 6
Unit 11
Object 1

Object 2
Unit 2
Unit 7
Unit 12
Object 2

Object 3
Unit 3
Unit 8
Unit 13
Object 3

Object 4
Unit 4
Unit 9
Unit 14
Object 4

Object 5
Unit 15
Unit 20
Unit 25
Object 5

Object 6
Unit 16
Unit 21
Unit 26
Object 6

Object 7
Unit 17
Unit 22
Unit 27
Object 7

Object 8
Unit 18
Unit 23
Unit 28
Object 8

Object 9
Unit 19
Unit 24
Unit 29
Object 9

Stripe Unit 4
Ceph striping

Object 0  Object 1  Object 2  Object 3  Object 4
Unit 0  Unit 1  Unit 2  Unit 3  Unit 4
Unit 5  Unit 6  Unit 7  Unit 8  Unit 9
Unit 10  Unit 11  Unit 12  Unit 13  Unit 14
Object 0  Object 1  Object 2  Object 3  Object 4

Strip Unit 4

Object 5  Object 6  Object 7  Object 8  Object 9
Unit 15  Unit 16  Unit 17  Unit 18  Unit 19
Unit 20  Unit 21  Unit 22  Unit 23  Unit 24
Unit 25  Unit 26  Unit 27  Unit 28  Unit 29
Object 5  Object 6  Object 7  Object 8  Object 9

Stripe 2
Ceph striping

Object 0

Object 0  Object 1  Object 2  Object 3  Object 4
Unit 0
Unit 5
Unit 10
Object 0

Object 1  Object 2  Object 3  Object 4
Unit 1
Unit 6
Unit 11
Object 1

Unit 2  Unit 7  Unit 12  Unit 13  Unit 4
Unit 3
Unit 8
Unit 14
Object 2

Unit 4
Object 3
Unit 9
Object 4

Object 5

Object 6
Unit 15
Unit 20
Unit 25
Object 5

Object 7
Unit 16
Unit 21
Unit 26
Object 6

Object 8
Unit 17
Unit 22
Unit 27
Object 7

Object 9
Unit 18
Unit 23
Unit 28
Object 8

Striped Unit 4

Striped 2
Ceph striping

Object 0

Object 1
Unit 1
Unit 6
Unit 11
Object 0

Object 2
Unit 2
Unit 7
Unit 12

Object 3
Unit 3
Unit 8
Unit 13

Object 4
Unit 4
Unit 9
Unit 14

Striped Unit 4

Object 5
Unit 15
Unit 20
Unit 25
Object 5

Object 6
Unit 16
Unit 21
Unit 26
Object 6

Object 7
Unit 17
Unit 22
Unit 27
Object 7

Object 8
Unit 18
Unit 23
Unit 28
Object 8

Object 9
Unit 19
Unit 24
Unit 29
Object 9

Disk 3
Ceph striping

Object 0

Object 1

Object 2

Object 3

Object 4

Unit 0

Unit 1

Unit 2

Unit 3

Unit 4

Unit 5

Unit 6

Unit 7

Unit 8

Unit 9

Unit 10

Unit 11

Unit 12

Unit 13

Unit 14

Unit 15

Unit 16

Unit 17

Unit 18

Unit 19

Unit 20

Unit 21

Unit 22

Unit 23

Unit 24

Unit 25

Unit 26

Unit 27

Unit 28

Unit 29

Object 5

Object 6

Object 7

Object 8

Object 9

Disk 3

Object Set 1

Stripe Unit 4

Stripe 2
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

Map processes data locally and returns key/value pairs.

Shuffle sends output to \textit{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
- close protoB
- stat protoB
- protoA

Steps:
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

A: t1
B: t2
B: t3
A: t4
A/protA/n1
A/protA/n2
B/protB/m1
B/protB/m2
Example: usage of protocols from logs

1. split logs on different nodes

```
open protoA
close
open protoB
close
stat
protoB
protoA
```

```
open protoA
close
open protoB
close
stat
protoB
protoA
```
Example: usage of protocols from logs

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

Diagram:
- open protoA
- close protoA
- open protoB
- close protoB
- stat protoB
- protoA
- protoB
- Map1
- Map2
- Map3
- Reduce1
- Reduce2
- A:t1
- B:t2
- A:t4
- B:t3
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

![Diagram showing the process of using protocols from logs with MapReduce](image)
The case of Hadoop, HDFS

HDFS: the Hadoop Distributed File System

- 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

Client

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4

InputFile

3 blocks

where to go?

1, 2, 3

where is file?

1: 2, 2: 4, 3: 1

OutputFile

3 blocks

Data storage and preservation
Parallel Read/Write in HDFS

InputFile \[\rightarrow\] 3 blocks \[\rightarrow\] Client \[\rightarrow\] NameNode

DataNode 1 \[\rightarrow\] DataNode 2 \[\rightarrow\] DataNode 3 \[\rightarrow\] DataNode 4
Parallel Read/Write in HDFS

InputFile \(\rightarrow\) 3 blocks \(\rightarrow\) Client \(\rightarrow\) NameNode

DataNode 1 \(\rightarrow\) DataNode 2 \(\rightarrow\) DataNode 3 \(\rightarrow\) DataNode 4

where to go? 1, 2, 3

1, 2, 3

where is file? 1: 2, 2: 4, 3: 1

OutputFile

InputFile

3 blocks

Client

where to go?

1, 2, 3

OutputFile

3 blocks

Parallel Read/Write in HDFS
Parallel Read/Write in HDFS

InputFile → 3 blocks → Client → NameNode

where to go? 1, 2, 3

DataNode 1

DataNode 2

DataNode 3

DataNode 4
Parallel Read/Write in HDFS

Client

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4
Parallel Read/Write in HDFS

Client

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4

InputFile

3 blocks

where to go?

1, 2, 3

OutputFile

3 blocks

where is file?

1: 2, 2: 4, 3: 1

1 3

2 1

3 2 1

3 2
Parallel Read/Write in HDFS

Client

where is file?
1:2, 2:4, 3:1

NameNode

DataNode 1
1 3

DataNode 2
2 1

DataNode 3
3 2 1

DataNode 4
3 2

InputFile
3 blocks

where to go?
1, 2, 3

OutputFile
3 blocks
Parallel Read/Write in HDFS

OutputFile → Client → NameNode

DataNode 1: 1, 3
DataNode 2: 2, 1
DataNode 3: 3, 2, 1
DataNode 4: 3, 2

InputFile: 3 blocks
OutputFile: 3 blocks

3 blocks where to go?
1, 2, 3

where is file?
1: 2, 2: 4, 3: 1
Parallel Map/Reduce

Client

JobTracker

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4
Parallel Map/Reduce

Client → JobTracker

1. DataNode 1
2. DataNode 2
3. DataNode 3
4. DataNode 4

.jar

Map 1
Map 2
Map 3
Reduce
Parallel Map/Reduce

Client

.jar

JobTracker

where?

1:2,2:4,3:1

NameNode

DataNode 1

1 3

DataNode 2

2 1

DataNode 3

3 2 1

DataNode 4

3 2
Parallel Map/Reduce

Client -> .jar -> JobTracker -> NameNode

Map 1: DataNode 1
Map 2: DataNode 3
Map 3: DataNode 2

1:2, 2:4, 3:1

where?

Mapreduce
Parallel Map/Reduce

Client

JobTracker

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4

Map 3

Map 1

Reduce

Map 2

Map 1: 2,2:4,3:1

Map 2: 3

Map 3: 3,1

Reduce: 3,2,1

Map 1

Map 2

Map 3

Reduce
Parallel Map/Reduce

Client

JobTracker

NameNode

DataNode 1

DataNode 2

DataNode 3

DataNode 4

Map 1

Map 2

Map 3

Reduce

1:2, 2:4, 3:1
Parallel Map/Reduce

1 3 4
DataNode 1

2 1 4
DataNode 2

3 2 1
DataNode 3

3 2 4
DataNode 4

Client

JobTracker

NameNode
Risks of data loss and corruption

1. Storage devices
2. Parallelizing files’ storage
3. Risks of data loss and corruption
4. Data consistency
5. Data safety
6. Conclusion
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
## 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

1. Storage devices
2. Parallelizing files’ storage
3. Risks of data loss and corruption
4. **Data consistency**
   - Checksums
   - Practical usage
5. Data safety
6. Conclusion
Checksum

**Definition**

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

\[ \begin{align*}
    & a_1 \quad a_2 \quad a_3 \quad a_4 \quad \cdots \quad a_i \quad \cdots \quad a_n \\
\end{align*} \]

\[ \text{CS} \]

\[ n \text{ blocks} \]

\[ b \quad \xleftrightarrow{\text{w}} \quad \text{CS} \]
Most basic checksum: data size

Computation

\[ a_1, a_2, a_3, a_4 \ldots a_i \ldots a_n \rightarrow n \]

\[ b = 8 \text{ bit} \quad w = 64 \text{ bit} \quad CS = n \]

Pros and Contra

- easy to compute
- detects erasures and additions
- does not detect any corruption
Basic checksum: sum/xor

**Computation**

\[ a_1 \ a_2 \ a_3 \ a_4 \ \ldots \ a_i \ \ldots \ a_n \rightarrow \sum a_i \]

\[ b \ \text{w} = b \quad CS = \sum_{i=1}^{n} a_i \]

**Pros and Contra**

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

Computation

\[ a_1 \ a_2 \ a_3 \ a_4 \ldots \ a_i \ldots \ a_n \rightarrow \sum a_i \sum ia_i \]

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>MB/s on intel core 2</th>
<th>Cycles Per Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adler32</td>
<td>920</td>
<td>1.9</td>
</tr>
<tr>
<td>MD5</td>
<td>255</td>
<td>6.8</td>
</tr>
<tr>
<td>SHA-1</td>
<td>153</td>
<td>11.4</td>
</tr>
<tr>
<td>SHA-256</td>
<td>111</td>
<td>15.8</td>
</tr>
<tr>
<td>SHA-512</td>
<td>99</td>
<td>17.7</td>
</tr>
</tbody>
</table>
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 storage and preservation

Data safety

1. Storage devices

2. Parallelizing files’ storage

3. Risks of data loss and corruption

4. Data consistency

5. Data safety
   - Redundancy
   - Parity
   - Erasure coding

6. Conclusion
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
Data storage and preservation

**Context**

- **k** pieces of data to be stored
- **n** nodes

\[
\begin{align*}
D_{1,2} & \quad D_{2,2} & \quad D_{3,2} \\
D_{1,1} & \quad D_{2,1} & \quad D_{3,1} \\
N_{1,1} & \quad N_{2,1} & \quad N_{3,1} \\
N_{1,2} & \quad N_{2,2} & \quad N_{3,2} \\
N_{1,3} & \quad N_{2,3} & \quad N_{3,3}
\end{align*}
\]

**Goal**

- Some nodes will fail/disappear
- We want our data back anyway
- We want the cheapest price
Data storage and preservation

Context

k pieces of data to be stored

\[ D_{1,1}, D_{2,1}, D_{3,1}, D_{1,2}, D_{2,2}, D_{3,2} \]

n nodes

\[ N_{1,1}, N_{1,2}, N_{1,3}, N_{2,1}, N_{2,2}, N_{2,3}, N_{3,1}, N_{3,2}, N_{3,3} \]

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

Disk 1  Disk 2  Disk 3

A  A  A
B  B  B
C  C  C
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**

Disk 1: A, B
Disk 2: A, C
Disk 3: A, B
Disk 4: B, C
Disk 5: B, C
Limitations of replication

With 2 replicas
- expansive: 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

- Disk 1: $A_1$, $B_1$, $C_1$
- Disk 2: $A_2$, $B_2$, $C_2$
- Disk 3: $A_3$, $B_3$, $C_3$
- Disk 4: $P_A$, $P_B$, $P_C$
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

Disk 1

- $A_1$
- $P_B$
- $C_3$

Disk 2

- $A_2$
- $B_1$
- $P_C$

Disk 3

- $A_3$
- $B_2$
- $C_1$

Disk 4

- $P_A$
- $B_3$
- $C_2$
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

\[ P = \sum D_i \quad Q = \sum g^i D_i \]

- 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

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

- 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

- Disk 1
  - $A_1$
  - $Q_B$
  - $P_C$

- Disk 2
  - $A_2$
  - $B_1$
  - $Q_C$

- Disk 3
  - $A_3$
  - $B_2$
  - $C_1$

- Disk 4
  - $P_A$
  - $B_3$
  - $C_2$

- Disk 5
  - $Q_A$
  - $P_B$
  - $C_3$
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 $n$

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

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

$$p_{n,i} = \binom{n}{i} p^i (1 - p)^{n-i}$$  \hspace{1cm} (3)
Graphically, for disks within a year

\[ p_{n,i} = 10^{-i} \]

- \( n = 16, p = 1\% \)
- \( n = 16, p = 3\% \)
- \( n = 32, p = 1\% \)
- \( n = 32, p = 3\% \)
- \( n = 48, p = 1\% \)
- \( n = 48, p = 3\% \)
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

\[ \begin{align*}
\text{k} &= 6 \\
\text{m} &= 2 \\
\text{n} &= 8 \\
\text{r} &= 4
\end{align*} \]
More generic example

Non systematic code

k=6 m=2 n=8 r=4
The matrix view

Generator Matrix $G$ is $nr \times kr$
The matrix view

Generator Matrix $G$ is $nr \times kr$
Back to systematic code

Systematic means $G$ includes identity
Practical example: RAID 6

Redundancy parity erasure
Systematic Reed-Solomon Codes

- 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)

$X = 1, 2, 3$

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

$k = 6$ and $n = 9$
Generic reconstruction

\[
\begin{array}{c}
d_1,1-d_1,r \\
d_2,1-d_2,r \\
d_3,1-d_3,r \\
d_4,1-d_4,r \\
d_5,1-d_5,r \\
d_6,1-d_6,r \\
\end{array}
\times
\begin{array}{c}
c_1,1-d_1,r \\
c_2,1-d_2,r \\
c_3,1-d_3,r \\
c_4,1-d_4,r \\
c_5,1-d_5,r \\
c_6,1-d_6,r \\
c_7,1-d_7,r \\
c_8,1-d_8,r \\
\end{array}
= 
\begin{array}{c}
c_1 \\
c_2 \\
c_3 \\
c_4 \\
c_5 \\
c_6 \\
c_7 \\
c_8 \\
\end{array}
\]
Generic reconstruction

\[
\begin{align*}
| & \quad d_8,1-d_8,r \\
\times & \quad d_7,1-d_7,r \\
| & \quad d_6,1-d_6,r \\
| & \quad d_5,1-d_5,r \\
| & \quad d_4,1-d_4,r \\
| & \quad d_3,1-d_3,r \\
| & \quad d_2,1-d_2,r \\
| & \quad d_1,1-d_1,r \\
\end{align*}
\]
Generic reconstruction

\[ \begin{array}{c}
\begin{array}{c}
\text{d}_{6,1}-d_{6,r} \\
\text{d}_{5,1}-d_{5,r} \\
\text{d}_{4,1}-d_{4,r} \\
\text{d}_{3,1}-d_{3,r} \\
\text{d}_{2,1}-d_{2,r} \\
\text{d}_{1,1}-d_{1,r}
\end{array}
\times
\end{array}
\begin{array}{c}
\rightarrow
\end{array}
\begin{array}{c}
\text{c}_{8,1}-d_{8,r} \\
\text{c}_{7,1}-d_{7,r} \\
\text{c}_{6,1}-d_{6,r} \\
\text{c}_{5,1}-d_{5,r} \\
\text{c}_{3,1}-d_{3,r} \\
\text{c}_{1,1}-d_{1,r}
\end{array}
\]

\( G_{\text{del}} \times D_{\text{del}} = C_{\text{del}} \)
Data storage and preservation

Generic reconstruction

\[ G_{del} \times D_{del} = C_{del} \]
Generic reconstruction

\[ G_{\text{del}} \times D_{\text{del}} = C_{\text{del}} \]

“Simple” matrix inversion

\[ D_{\text{del}} = G_{\text{del}}^{-1} \times C_{\text{del}} \]
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

1. Storage devices
2. Parallelizing files’ storage
3. Risks of data loss and corruption
4. Data consistency
5. Data safety
6. 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