Have a Seat on the ErasureBench: Easy Evaluation of Erasure Coding Libraries for Distributed Storage Systems

2016 Workshop on Planetary-Scale Distributed Systems

Sébastien Vaucher, Hugues Mercier, Valerio Schiavoni

Institute of Computer Science
Université de Neuchâtel, Switzerland

sebastien.vaucher@unine.ch

Budapest, Hungary, 26 September 2016

Motivation

More and more data needs to be stored reliably on online servers.
Reliability can be provided through:

- Replication
- Erasure coding

Motivation

More and more data needs to be stored reliably on online servers.
Reliability can be provided through:

- Replication
- Erasure coding

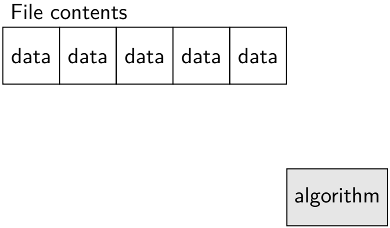
Motivation

The characteristics of an erasure coding algorithm are difficult to evaluate.
Evaluation is often done theoretically or by simulation.

Erasure coding

Goal: add redundancy to cope with data loss/corruption

Example using a $(5, 2)$ Reed-Solomon code:



File contents

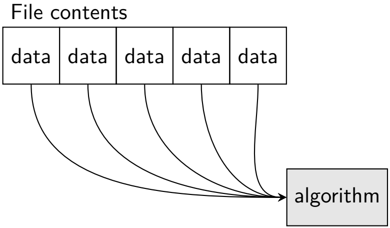
data data data data data

algorithm

Erasure coding

Goal: add redundancy to cope with data loss/corruption

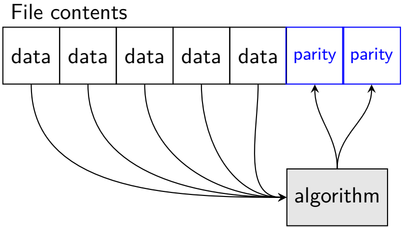
Example using a $(5, 2)$ Reed-Solomon code:



Erasure coding

Goal: add redundancy to cope with data loss/corruption

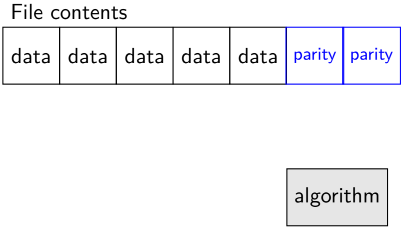
Example using a $(5,2)$ Reed-Solomon code:



Erasure coding

Goal: add redundancy to cope with data loss/corruption

Example using a $(5, 2)$ Reed-Solomon code:



File contents

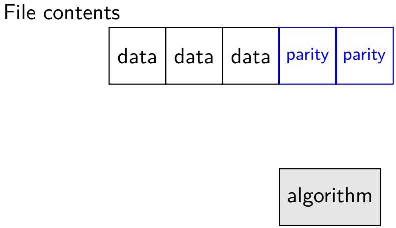
data data data data data parity parity

algorithm

Erasure coding

Goal: add redundancy to cope with data loss/corruption

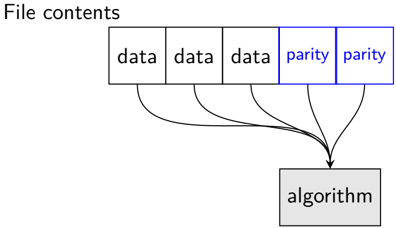
Example using a $(5, 2)$ Reed-Solomon code:



Erasure coding

Goal: add redundancy to cope with data loss/corruption

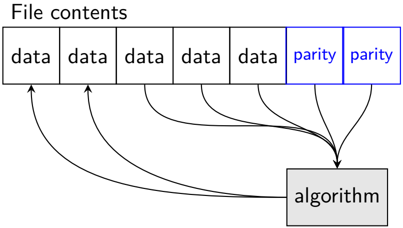
Example using a $(5, 2)$ Reed-Solomon code:



Erasure coding

Goal: add redundancy to cope with data loss/corruption

Example using a $(5, 2)$ Reed-Solomon code:



Key features

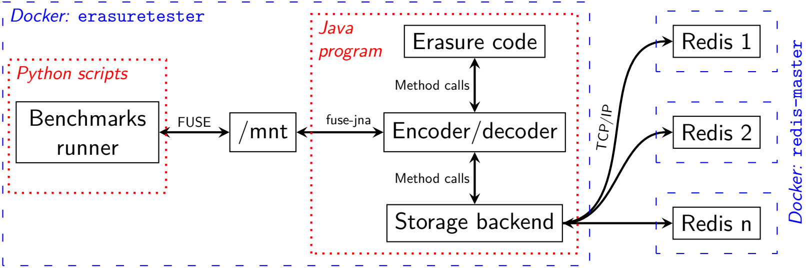
- Compatible with existing benchmark programs
- Automated benchmarks execution
- Containerized storage nodes (> 1 per physical node)
- Replay fault traces

Evaluation example

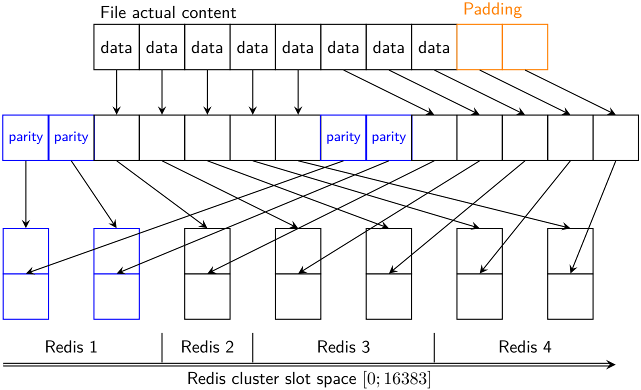
How to evaluate a new algorithm.

1. Program the algorithm as a Java class
2. Write benchmarks as Python functions
   - Debian-compatible programs can be launched as sub-processes
3. Configure the evaluation
   - e.g. algorithm parameters, fault trace, ...
4. Easily deploy the solution to a Docker cluster
5. Collect results

Technical components



Blocks distribution



Metadata management

Each block is identified by a 32-bit key. Using it, we derive:

1. Key of the blocks aggregation stored in Redis
2. Offset within that aggregation

The list of all block keys is kept in memory.

Automated deployment and scaling

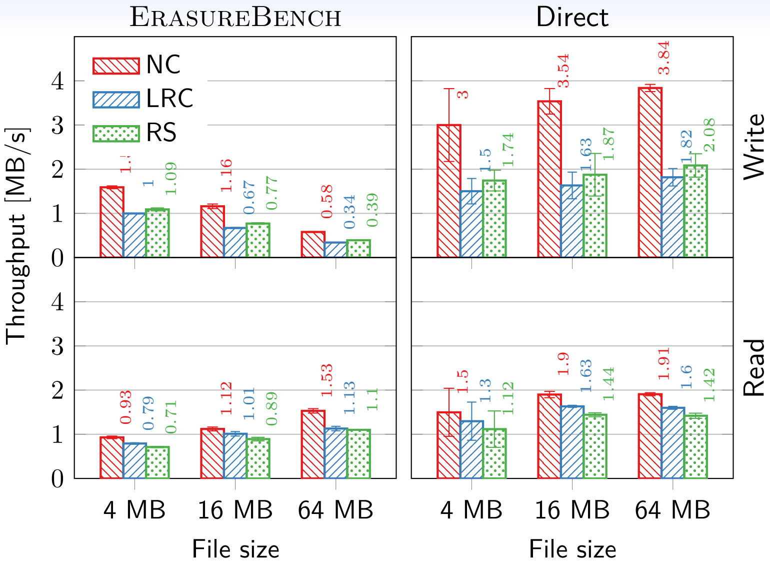
As part of ErasureBench, we provide scripts that automate the deployment of the solution to a Docker Swarm cluster, up to the collection of results.

Evaluation

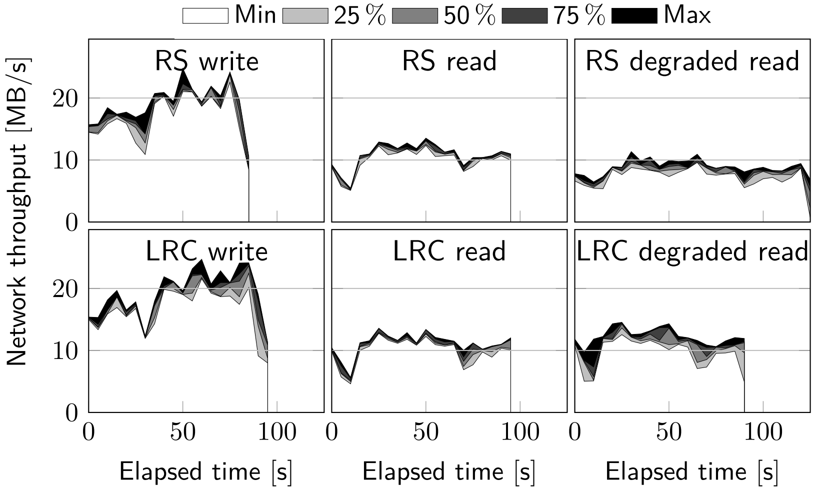
We evaluated algorithms from ”XORing Elephants: Novel Erasure Codes for Big Data”.

NC	No erasure coding
RS	Reed-Solomon $(10, 4)$
LRC	Locally Repairable Code $(10, 6, 5)$

Throughput

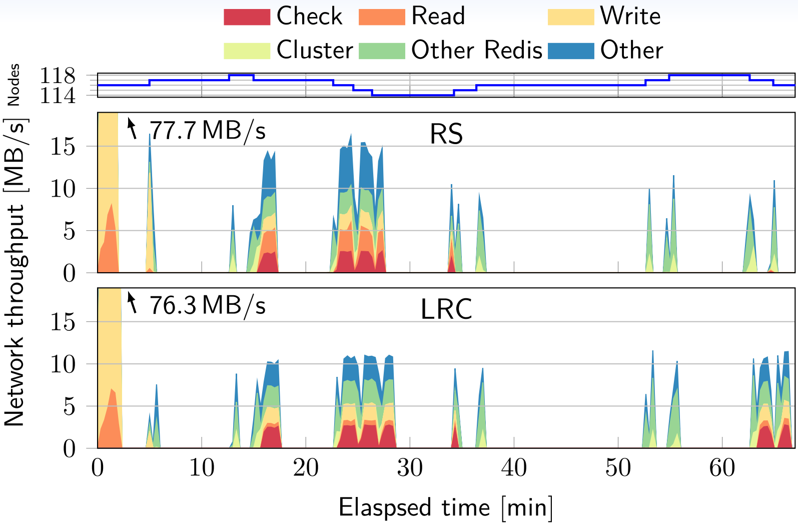

ErasureBench
Direct
NC
LRC
RS
Throughput [MB/s]
0
1
2
3
4
1.
1
1.09
1.16
0.67
0.77
0.58
0.34
0.39
3
1.5
1.74
3.54
1.63
1.87
3.84
1.82
2.08
Write
0.93
0.79
0.71
1.12
1.01
0.89
1.53
1.13
1.1
1.5
1.3
1.12
1.9
1.63
1.44
1.91
1.6
1.42
Read
4 MB
16 MB
64 MB
File size

Traffic



Min
25 %
50 %
75 %
Max
RS write
RS read
RS degraded read
LRC write
LRC read
LRC degraded read
Network throughput [MB/s]
0
10
20
Elapsed time [s]
0
50
100

Trace


Check
Read
Write
Cluster
Other Redis
Other
Nodes
118
114
Network throughput [MB/s]
77.7 MB/s
RS
76.3 MB/s
LRC
0
5
10
15
0
10
20
30
40
50
60
Elaspsed time [min]

Conclusion

Using ErasureBench, evaluating an erasure coding algorithm under real conditions is easier and cheaper.

Available open-source at
`https://github.com/safecloud-project/erasurebench`