Exercise 2 : checksums, RAID and erasure coding

Sébastien Ponce

May 22, 2015

1 Goals of the exercise

- Play with checksums and compare efficiency and robustness
- Use hand written version of RAID systems and test error recovery
  - with striping
  - with mirroring
  - with parity
- Test erasure coding

2 Setup

2.1 Basics

- open a terminal
- go to the exercise2 directory

```
cd /data/io/exercise2
```

2.2 Create data files

- Use the generatePatternFile.py in the tools directory to create a data file with a readable pattern.

```
python ../tools/generatePatternFile.py patternFile
```

- check the content of the pattern file : numbers from 0 to 9999

```
less patternFile
```

- Use the generateTemperatureFile.py in the tools directory to create a data file. Use 10000 captors and 100 points for a 40MB file.

```
python ../tools/generateTemperatureFile.py 10000 100 datafile
```

- Be patient, it should take around 2mn
- check that the file is in the kernel cache

```
linux-fincore datafile
```

3 Checksums

Go to checksum subdirectory

```
cd checksum
```

3.1 computation speed

- compute xor32, adler32, md5, sha1 and sha256 checksums for your datafile from memory (as it is in cache)

```
./xor32.py ../datafile
./adler32.py ../datafile
./md5.py ../datafile
./sha1.py ../datafile
./sha256.py ../datafile
```

- compare timings except for xor that was implemented in python manually
- note by the way the difference of speed between non optimized python and optimized compiled code !

3.2 error detection

- change first byte of the input file (remember previous value ! it's 0x10)

```
hexdump -C -n 16 ../datafile
printf '\x00' | dd of=../datafile bs=1 count=1 conv=notrunc
hexdump -C -n 16 ../datafile
```

- Recompute all checksums and see the changes in the different algorithms

```
./xor32.py ../datafile
./adler32.py ../datafile
./md5.py ../datafile
./sha1.py ../datafile
./sha256.py ../datafile
```

- note that xor changes only by one bit and adler has its lower part almost unchanged (again only by one bit). Note how the other ones are completely different although we changed only one bit of input.
- put back proper first byte (was 0x10) but exchange the first 4 bytes with the nest 4. Take care that the next 4 are specific to your file ! Put the right values in the following code

```
hexdump -C -n 16 ../datafile
printf '\x73\x5d\xfc\x40\x10\x27\x00\x00' | \
  dd of=../datafile bs=1 count=8 conv=notrunc
hexdump -C -n 16 ../datafile
```

- Recompute all checksums and see the changes in the different algorithms

```
./xor32.py ../datafile
./adler32.py ../datafile
./md5.py ../datafile
./sha1.py ../datafile
./sha256.py ../datafile
```

- note in particular that xor does not detect exchanges and that adler32 has same lower part.
- repair file so that it's readable. Again, put the right bytes for your case !

```
hexdump -C -n 16 ../datafile
printf '\x10\x27\x00\x00\x73\x5d\xfc\x40' | \
  dd of=../datafile bs=1 count=8 conv=notrunc
hexdump -C -n 16 ../datafile
```

4 RAID levels

4.1 Introduction

We will play here with fake RAID systems, where disks were replaced by directories. The implementations are done in python and are highly inefficient.

The only point of this exercise is to have simple code that you can understand eaasily and demonstrates how RAID technologies work.

4.2 RAID 0 : striping

- Go to RAID0 subdirectory

```
cd ../RAID0
```

- create a RAID setup using the createRAID0 script. Check its code

```
./createRAID0.py 5
```

- store the pattern file into the RAID array using copyToRAID0.py. Check its code

```
./copyToRAID0.py ../patternFile
```

- have a look at the content of the striped

```
less Disk0/patternFile.0
less Disk1/patternFile.1
less Disk2/patternFile.2
```

- read back the file using readFromRAID0 and check consistency

```
./readFromRAID0.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

4.3 RAID 1 : mirroring

- Go to RAID1 subdirectory

```
cd ../RAID1
```

- create a RAID setup using the createRAID1 script. Check its code

```
./createRAID1.py 3
```

- store a file into the RAID array using copyToRAID1.py. Check its code

```
./copyToRAID1.py ../patternFile
```

- check how file was replicated

```
ls -al Disk*
```

- read back the file and check consistency. Check the read back code

```
./readFromRAID1.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- suppress one copy and check you can still read

```
rm Disk0/patternFile
./readFromRAID1.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- corrupt another copy and check you can still read. Note the use of checksums

```
vim Disk1/patternFile
./readFromRAID1.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- ask for a rebuild of the RAID system and check effect

```
../checksum/md5.py ../patternFile
ls -l Disk*
../checksum/md5.py Disk1/patternFile
../checksum/md5.py Disk2/patternFile
./repairFileInRAID1.py patternFile
ls -l Disk*
../checksum/md5.py Disk0/patternFile
```

- try corrupting a file or its checksum and see that you can read back and repair

```
vim Disk1/patternFile.1
./readFromRAID1.py patternFile patternFileCopy
../checksum/md5.py patternFile
../checksum/md5.py patternFileCopy
./repairFileInRAID1.py patternFile
```

4.4 RAID 5 : parity

- Go to RAID5 subdirectory

```
cd ../RAID5
```

- create a RAID setup using the createRAID5 script. Check its code

```
./createRAID5.py 4
```

- store a file into the RAID array using copyToRAID5.py. Check its code

```
./copyToRAID5.py ../patternFile
```

- check how file was split and how parity was added

```
ls -al Disk*
less Disk0/patternFile.0
less Disk2/patternFile.2
less Disk3/patternFile.3
```

- read back the file and check consistency. Check the read back code

```
./readFromRAID5.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- check storage overhead

```
ls -l ../patternFile
du -cb Disk*/pattern*
```

- suppress one stripe and check you can still read

```
rm Disk0/patternFile.0
./readFromRAID5.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- ask for a rebuild of the RAID system and check effect

```
ls -l Disk*
./repairFileInRAID5.py patternFile
ls -l Disk*
```

- corrupt a stripe or its checksum and check you can still read

```
vim Disk1/patternFile.1
./readFromRAID5.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
./repairFileInRAID5.py patternFile
```

5 erasure coding

We use here an external library to provide erasure coding code. It's called zfec and has a nice and simple python API (see https://pypi.python.org/pypi/zfec and go to "Python API").

- Go to erasure subdirectory

```
cd ../erasure
```

- create a erasure coded pool with 3 + 4 disks using the createErasure script. Check its code

```
./createErasure.py 3 4
```

- store a file into the erasure coded pool using copyToErasure.py. Check its code

```
./copyToErasure.py ../patternFile
```

- check how file was split and how extra chunks were added

```
ls -al Disk*
less Disk0/patternFile.0
less Disk2/patternFile.2
less Disk5/patternFile.5
```

- read back the file and check consistency

```
./readFromErasure.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- check storage overhead

```
ls -l ../patternFile
du -cb Disk*/pattern*
```

- suppress one stripe and check you can still read

```
rm Disk0/patternFile.0
./readFromErasure.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- corrupt another stripe and check you can still read

```
vim Disk1/patternFile.1
./readFromErasure.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- go for one more deletion and one more corruption. Check you can still read !

```
vim Disk4/patternFile.4
rm Disk5/patternFile.5
./readFromErasure.py patternFile patternFileCopy
../checksum/md5.py ../patternFile
../checksum/md5.py patternFileCopy
```

- ask for a rebuild of the erasure coded pool and check effect

```
ls -l Disk*
./repairFileInErasure.py patternFile
ls -l Disk*
```

- have a look at the readFromErasure code and see how all this happens