Checksum Support

EOS WORKSHOP

Andreas-Joachim Peters
CERN IT Storage Group

Overview

- checksum types
- improvements in EOS v4.6.8
  - benchmarks
- impact on IO / outlook

Checksum Support

- **Classical EOS checksum types**
  - **adler32**
  - **crc32**
  - **crc32c**
  - **md5**
  - **sha1**

Classical checksum usage

- adler32: file checksum in **all LHC VOs** but **ALICE**
- md5: file checksum in **ALICE**
- md5: file checksum for **S3** storage
- crc32: **zip** library / *.root files
- crc32c: **scsi - default block checksum in EOS rain**

AVX accelerated

adler & crc checksums are non-cryptographic hashes which can be combined (= parallelized)

md5 computation cannot be parallized!

IO impact of checksumming

During EC testing for ALICE we realised that the **single stream performance is bottlenecked** mainly by **MD5 computation**.

Started investigation to get faster implementation of currently supported checksumming algorithms and added some modern ones and a generic **eos-checksum** command, which provides all available flavours.

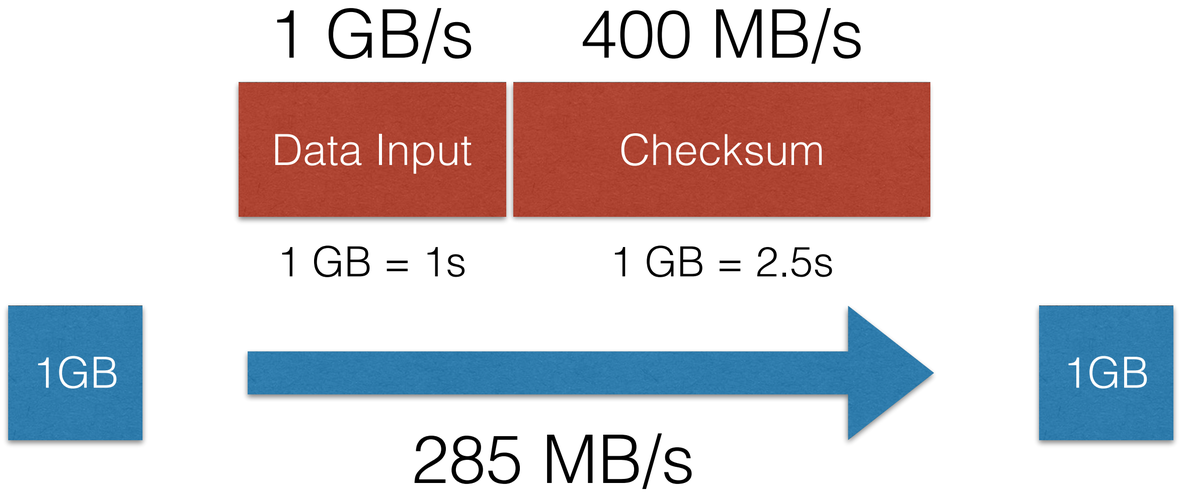
Intel-ISAL(-crypto) libraries bring AVX accelerated version of several non-cryptographic checksums for EL7.

‘Unfortunately’ the **currently used MD5** implementation (openssl) **is** the **fastest available.**

IO impact of checksumming

slow checksum

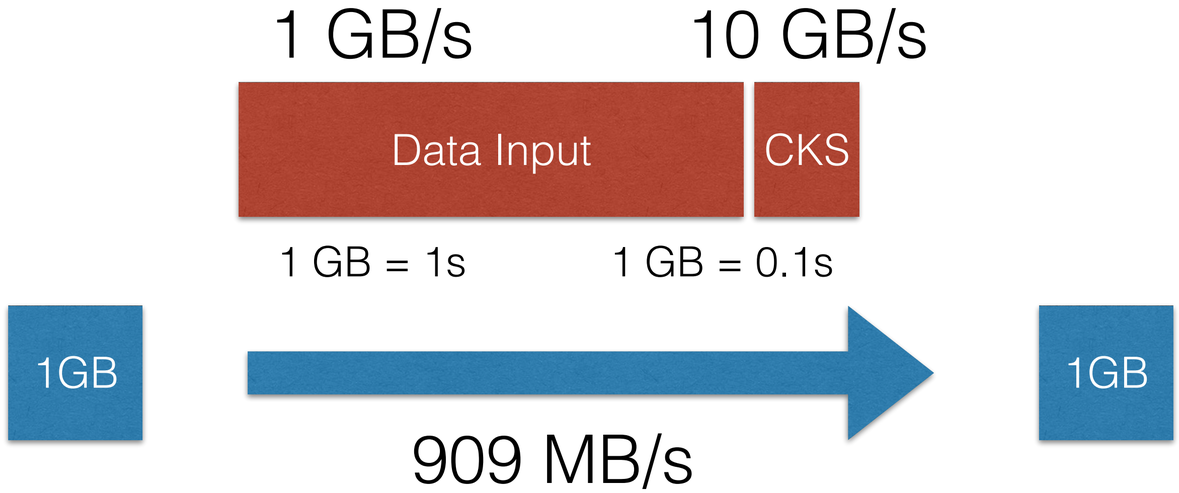
sync. pipeline



IO impact of checksumming

fast checksum

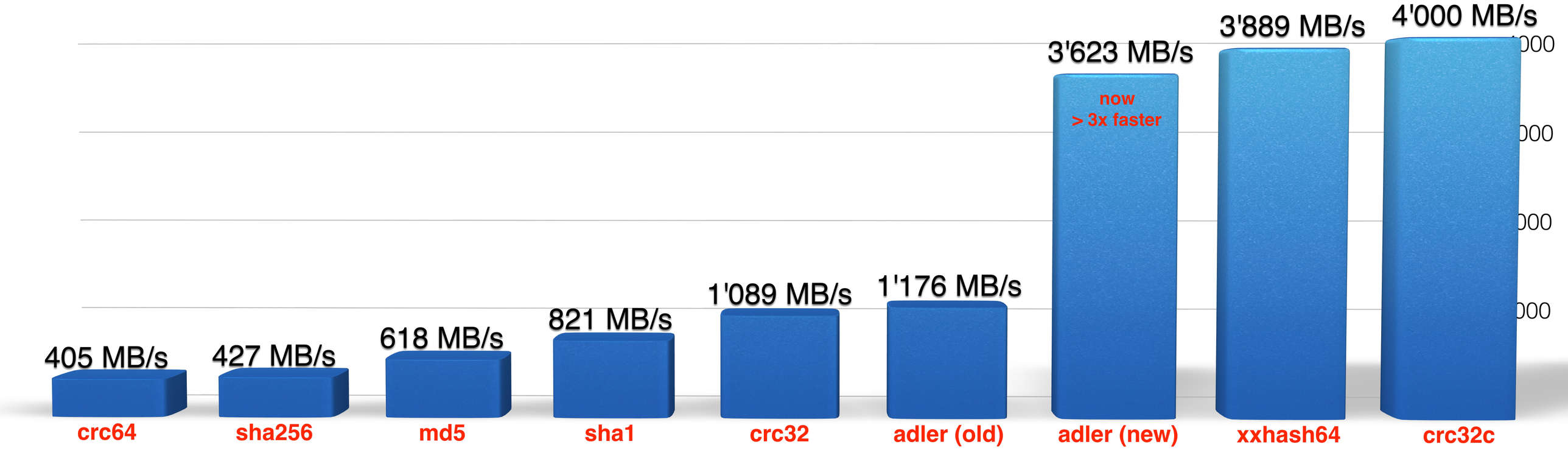
sync. pipeline



EOS 4.6.8

Checksum Benchmarks

eos-checksum <flavor> <cached-file> @ 2.3GHz Xeon



remark: this is not the speed of the algorithm alone, but IO + algorithm!

Summary

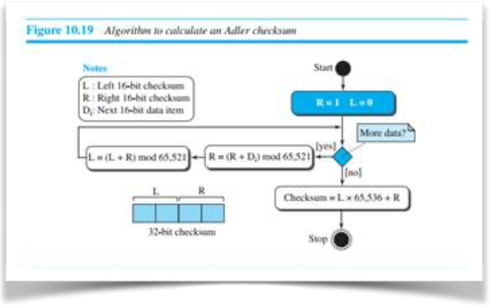


Figure 10.19 Algorithm to calculate an Adler checksum

Notes
L : Left 16-bit checksum
R : Right 16-bit checksum
D_i: Next 16-bit data item

Start
R = 1 L = 0
More data?
[yes]
$R = (R + D_i) \bmod 65,521$
$L = (L + R) \bmod 65,521$
[no]
Checksum = L × 65,536 + R
Stop
L R
32-bit checksum

We have modernised the stack of supported checksums and provide now the fastest available implementations for adler & crc32c since EOS 4.6.8

If you have to chose: avoid cryptographic checksums if they are not required to identify contents. *adler* & *crc* flavours can combine hashes of blocks into final hashes and are compatible with the concepts of distributed storage, where a file is not only located on a single disk.

We have seen in incidents at CERN that adler32 is actually not good enough to identify certain systematic hardware bit corruptions. It is simple to construct corruptions with identical adler values.