# XRootD pgRead & pgWrite

XRootD Workshop
March 29-31, 2023

Andrew Hanushevsky, SLAC
http://xrootd.org

# Page read/write (pgRead/pgWrite)

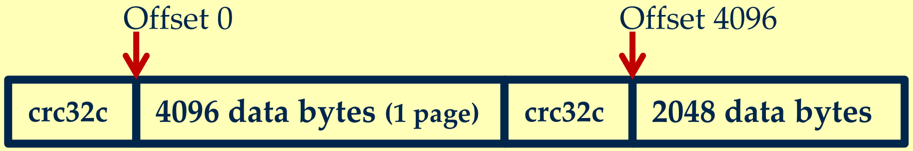
- These are page aligned reads/writes
  - 4K pages on 4K boundaries
    - Does allow misalignment for 1st page (later)
  - Each page is check summed using crc32c
    - Follows IETF RFC 7143 standard
  - Client/server perform on-the-fly correction
    - Reads: client rereads pages in error
    - Writes: server supplies pages in error to rewrite

# Why page read/write

- Transmission errors do occur
  - Some not caught by the TCP 16 bit checksum
    - Reports of errors on some international links
      - Typically during high usage periods
  - Avoids retransmission of large files (> 10GB)
    - When only a few bits are corrupted
  - Avoids having sticky errors in Xcache
    - A serious concern in a long-lived page cache

# Wire layout

**Read/write 6144 bytes at offset 0** (page aligned - typical xrdcp, Xcache)

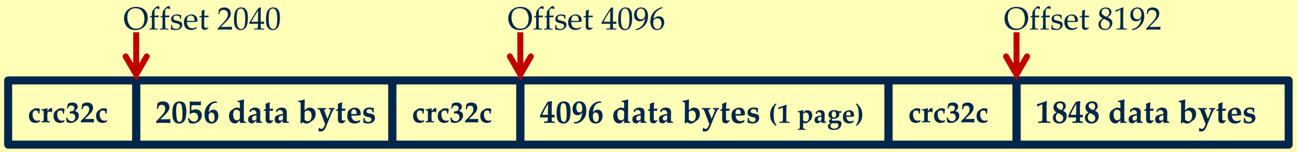


Offset 0

Offset 4096

crc32c | 4096 data bytes (1 page) | crc32c | 2048 data bytes

**Read/write 8000 bytes at offset 2040** (non-page aligned - typical random I/O)



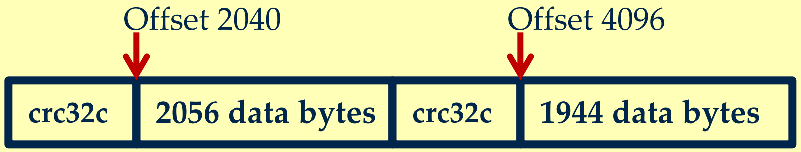
Offset 2040

Offset 4096

Offset 8192

crc32c | 2056 data bytes | crc32c | 4096 data bytes (1 page) | crc32c | 1848 data bytes

**Read/write 4000 bytes at offset 2040** (non-page aligned – degenerate case)



Offset 2040

Offset 4096

crc32c | 2056 data bytes | crc32c | 1944 data bytes

# Special Server Response

- Page Read/Write use a new response type
  - kXR_status
    - Response header is check summed using crc32c
      - Also provides extended contextual data
        - Minimizes need for client to maintain state
    - Response data is check summed using crc32c
      - For pgWrite final response data lists pages in error
        - Client should retransmit these pages
      - Server maintains list of uncorrected pages
        - Maximum of 256 pages may be left uncorrected

# Page read/write sync vs. async

- Checksum processing restricts I/O size
  - Sync: 2,093,056 max bytes per I/O seg
    - Accounts for checksum overhead
      - Data + checksums ~= 2 MB (max default buffer size)
        - 2093056/4096 = 511
        - 511*4+2093056 = 2095100
          - 52 bytes shy of 2MB
  - Async: 64K per I/O segment
    - Sweet spot to minimize latency
  - Values cannot be adjusted

# Final Notes on Async I/O

- Async only enabled for networked devices
  - Linux async I/O useless for locally attached disk
    - Implemented at user level via threads
- May change with new io_uring  interface
  - Available since Linux Kernel version 5.1
    - RH 8.7 uses 4.18
    - RH 9.1 uses 5.14 (yay!)
- Adoption rates push this 1 to 2 years hence

# FAQ I

- Why crc32c?
  - Excellent for bit error detection
    - Random not systematic (i.e. hacked data)
      - Systematic detection needs a cryptographic checksum
  - Hardware assisted (Intel & AMD)
    - Can compute checksum up to 8 bytes/cycle
      - Note ARM implements CRC32
  - Used by modern (and not so modern) systems
    - iSCSI, gcs, Btrfs, ext4, Ceph, among others

# FAQ II

- Why 4K page size?
  - Good fit for crc32c to maximize error detection
  - Good for transitive checksum processing
    - Specific to XrdOssCsi plug-in
      - Provides checksum protection for data on disk (like zfs)
      - Avoids having to recalculate checksum
      - Good page size for disk based files
  - Chosen to avoid page size zoo
    - Would be a mess if multiple page sizes allowed

# FAQ III

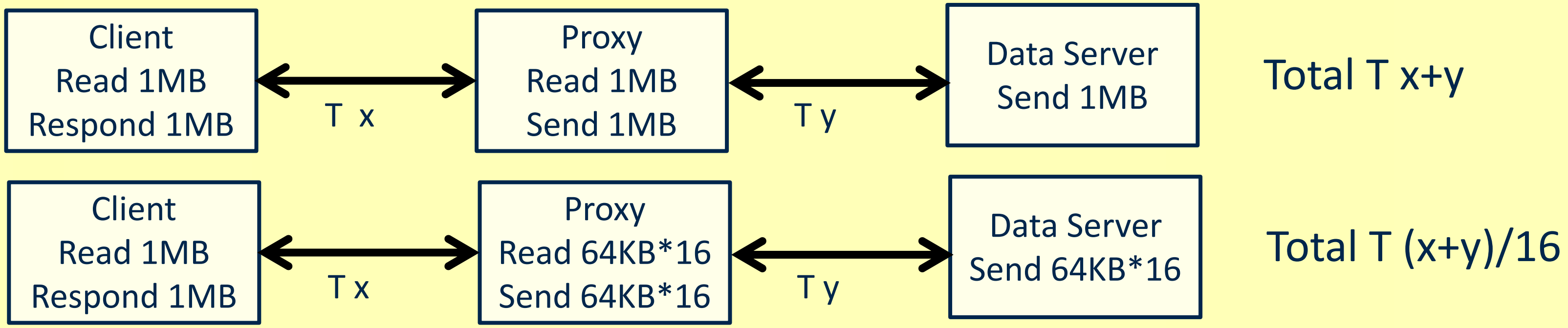
- What if a server doesn’t support pgXXX?
  - Client reverts to using TLS if possible
    - TLS closes the connection upon checksum error
    - Client can recover at this point
      - For reads, reconnects and rereads
      - For writes, reconnects and rewrites
        - But must rewrite more data than needed
  - If TLS not available, uses normal read/write
    - This is configurable for Xcache
      - See the **pfc.cschk** directive

# FAQ IV

- What happens if file closed with errors?
  - If client has not corrected all errors upon close
    - Server writes to log that the file is corrupted
    - The close fails
      - Assumption is that client will use POSC upon open()
        - Since the close failed the file gets deleted
    - We are still looking for enhancement suggestions
      - What would be a better approach w/o duplication?

# FAQ V

- Why 64K async size?
  - Minimize store/Forward effect in proxy servers
    - This also includes Xcache



Client
Read 1MB
Respond 1MB

T x

Proxy
Read 1MB
Send 1MB

T y

Data Server
Send 1MB

Total T x+y

Client
Read 1MB
Respond 1MB

T x

Proxy
Read 64KB*16
Send 64KB*16

T y

Data Server
Send 64KB*16

Total T (x+y)/16

- Chunking a read keeps the pipe full
  - Almost streaming but at a lower CPU cost
    - Aggregate performance can be achieved

# FAQ VI

- Why is async size not configurable?
  - Addition of checksum complicates things
    - Non-standard buffer sizes create headaches
      - Sometimes need to be oddly aligned
      - Sometimes not fully utilized
  - Since 64K is the WAN sweet spot
    - We decided to standardize on that size
      - Note TLS is already standardized on this buffer size
        - No one seems to be complaining about that

# Conclusion

- XRootD pgRead/Write is a game changer
  - Provides integrity for data in motion
    - Low cost for computation & recoverability
      - Integrated with integrity for data at rest (XrdOssCsi)
- Our core partners
  - CERN SLAC UCSD
- Community & funding partners *(not a complete list)*
  - OSG WLCG ATLAS EXPERIMENT ALICE CMS LSST dCache GSI DOE NSF

Funding from US Department of Energy contract DE-AC02-76SF00515 with Stanford University