

# XCache for streaming data workflows

Nikolai Hartmann, Guenter Duceck, Christoph Anton Mitterer, Rodney Walker

LMU Munich

May 11, 2021, IDT-UM collaboration meeting



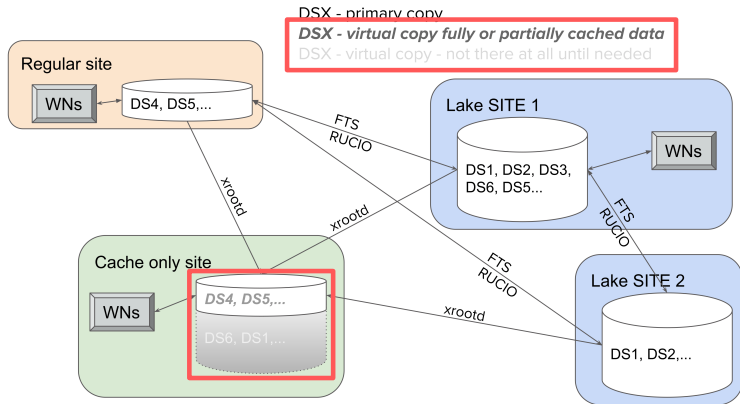
# What is XCache?

- Disk caching proxy using xrootd (`libXrdFileCache.so`)
- Data is cached in blocks  
→ suitable for streaming data, direct I/O
- Simply prepend xcache server url - e.g.  
`TFile::Open("root:[xcache-server]:[port]//[origin-xrootd-path]")`
- Optionally use rucio DIDs via N2N plugin:  
<https://github.com/xrootd/rucioN2N-for-Xcache>  
→ allows usage of rucio DIDs instead of xrootd path  
→ tracks identical files distributed at different locations  
(internal symlink `.../scope/XX/YY/filename`)

# Munich XCache Setup

- Hardware: Old dCache pool node (from 2012):
  - Dell R710, 2x6 core Xeon L5640, 32 GB RAM, 10 Gb Ethernet
  - 60 TB Raid (2x12x3TB HDD)
    - now operated as individual disks
- Second node with similar Hardware for testing new versions/setups
  - also want to test “cluster” of XCaches with redirector
  - alternative: scheduling of jobs to load data from individual XCaches
- Xrootd version 5.1.0
- Setup using singularity centos7 image. Full configuration:  
<https://gitlab.physik.uni-muenchen.de/Nikolai.Hartmann/xcache-singularity-lrz/>
- XCache settings:  
`pfc.ram 14g`  
`pfc.blocksize 1M`  
`pfc.prefetch 0`
  - no prefetch for streaming data workflows
  - also experimenting with smaller block sizes

# Virtual placement at ATLAS



(Graphic by Ilija Vukotic)

- Virtually place datasets to cache-only sites
  - XCache is registered as a rucio storage element, but data is not actively transferred
  - jobs get scheduled to sites with virtually placed data
- Expected to ensure high hit rates
- Currently being tested at various sites in ATLAS
  - Running an analysis job queue that uses our XCache server

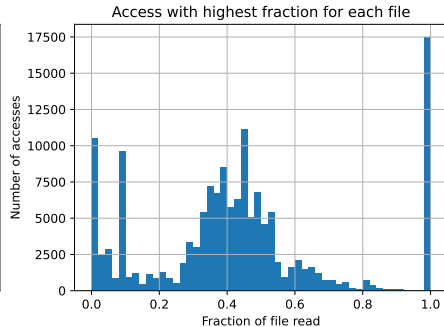
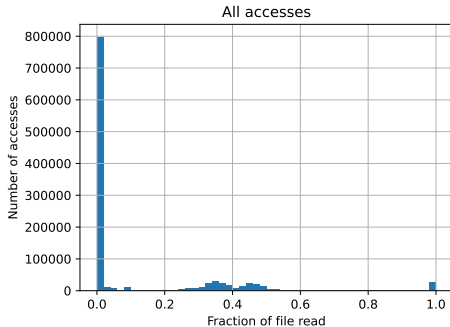
# Monitoring of accesses in XCache

- Access statistics are (currently) tracked in `.cinfo` files (binary format)
- New entry written after file is closed
- Tracked information per access:
  - Attach/Detach time: time between open/close
  - Bytes hit: Read from cache
  - Bytes missed: Read from remote (and placed in cache)
  - Bytes bypassed: Passed through from remote (and not placed in cache)
- Tracked information per file: (among others)
  - File size
  - Block size
  - Blocks cached (bit mask)
- Collected in a ES database at UChicago for ATLAS VP sites
  - currently directly read from `.cinfo` files
  - testing new g-stream monitoring in xrootd 5.1

# Sparse reading of files

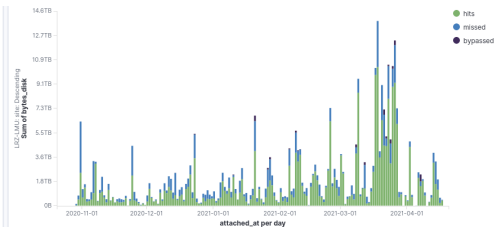
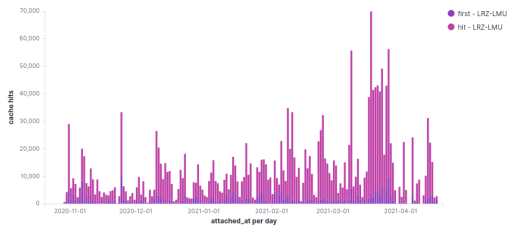
Analysis jobs typically access only a certain fraction of the data  
(typically a fraction of all branches in a ROOT file)

Data from last 2 months on our XCache analysis queue:

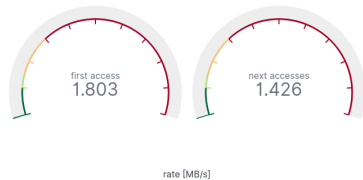


# Hit rate from XCache monitoring

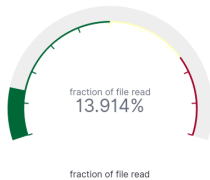
From ES Chicago database



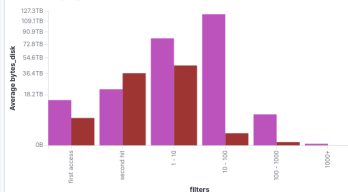
xcache - average rate



xcache - fraction read



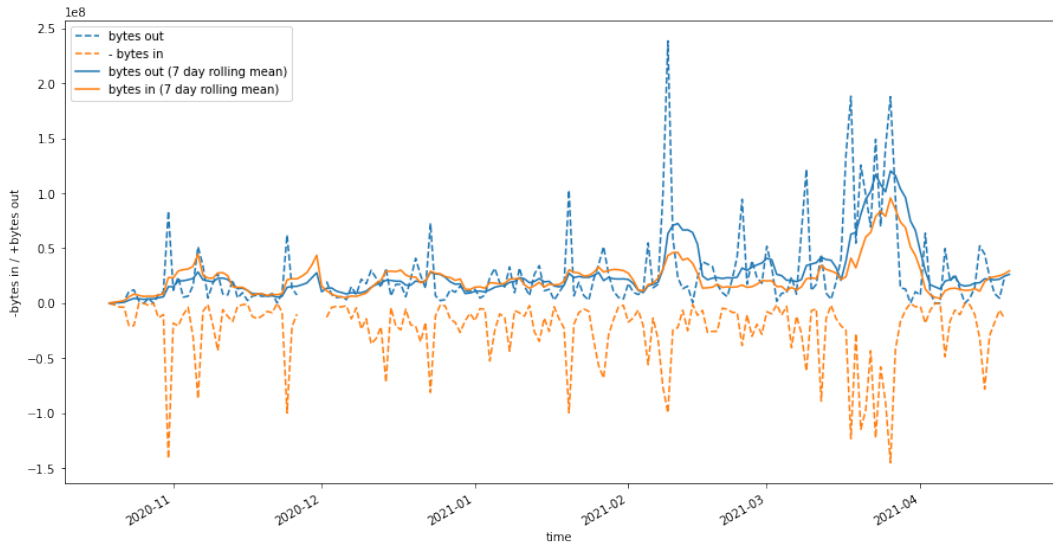
xcache - delivery origin



→ would expect much higher amount of output data than input data for cache

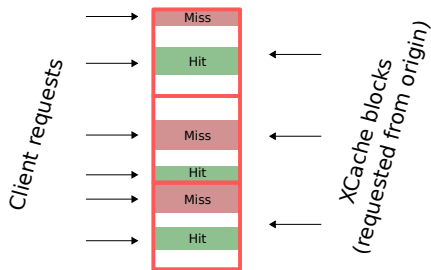
# Hit rate in terms of data input/output

From Ganglia server monitoring



→ In reality rather balanced (recently a bit higher output)

## But why?



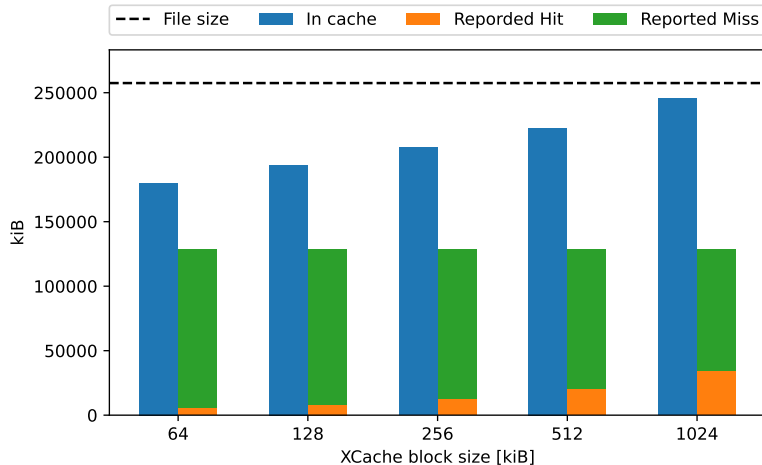
Main effects:

- Fetching of blocks and serving requests happens asynchronously
  - request might count as "served from cache" although download was actually triggered by the same job
  - similar situation for requested chunks that span multiple (xcache) blocks or the other way round
  - happens also when downloading larger chunks (e.g. xrdcp) without vector reads because xrootd will split them up into smaller ones that are asynchronously served to the client
- Sparse reading with small basket sizes in ROOT and large block sizes in XCache could lead to larger amount being downloaded

→ **What is monitored as "hit rate" is not what we are mainly interested in ...**

# Try a smaller block size?

SUSY analysis example, running on a ttbar MC ROOT file, first access:



Previous setting: 1024 kiB

Currently testing: 128 kiB

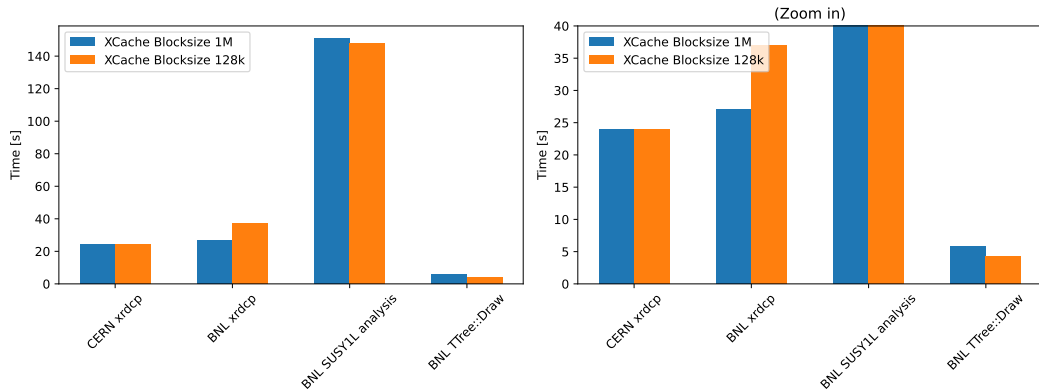
# Impact of latency with smaller block size

XCache block size determines chunk sizes for reading from remote

→ might impact performance due to latency (but chunks are requested async)

For a 249MB DAOD (on my laptop in Munich with ~80Mbit/s)

Latencies: ~30ms (CERN), ~100ms (BNL)



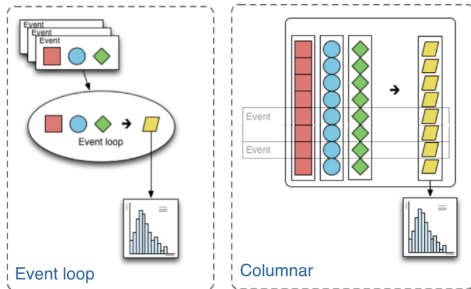
→ some performance degradation for full file transfers with high latency (100ms)

→ benefits from smaller amount of transferred data dominate for sparse access

# Conclusions

- Testing virtual placement system at ATLAS
  - XCache included as “virtual” storage element
- Blockwise caching can support streaming data workflows with sparse access
  - sparse reading common for analysis workflows
- Monitoring of hit rates more difficult
  - does not necessary coincide with total input/output
- Smaller block sizes could help increase the caching efficiency for sparse access
  - currently testing block size of 128kiB instead of 1MiB
- Outlook: we are purchasing an SSD based cache server ( $\approx 50$  TB) which should give better performance and support much higher loads.

# Columnar data analysis



- We investigated also in some detail columnar data analysis
  - important for data access and caching (different access patterns)
  - see [my presentation on the last IDT-UM meeting](#)
- However, many analysis concepts still to be developed
  - outside of scope of ErUM/IDT Caching studies
- But we contribute to these important developments
  - started developing columnar data analysis model for ATLAS DAOD\_PHYSLITE
  - [vCHEP presentation next Wednesday](#)

<sup>1</sup>Plot from <https://coffeateam.github.io/coffea/concepts.html>

# Backup

## Basket sizes (example for a ATLAS DAOD)

