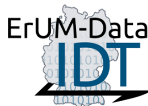
XCache for grid computing and analysis

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

LMU Munich

February 14, 2022, IDT-UM collaboration meeting



What is XCache?

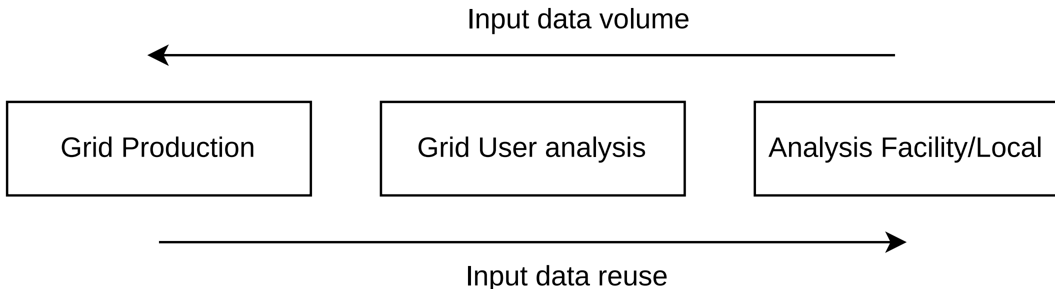
- Disk caching proxy using xrootd (`libXrdFileCache.so`)
- Data is cached in blocks
  → suitable for streaming data, direct I/O
- Simply prepend xcache server url - e.g.
  `TFile::Open("root:[xcache-server]:[port]//[origin-xrootd-path]")`
- Optionally use rucio DIDs via N2N plugin:
  https://github.com/xrootd/rucioN2N-for-Xcache
  → 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: 2 Old dCache pool nodes:
  - Dell R710, 2x6 core Xeon L5640, 32 GB RAM, 10 Gbps Ethernet
  - 60 TB Raid (2x12x3TB HDD)
    → now operated as **individual disks**
    → **replaced by newer ones (80 TB, 64GB RAM each) beginning of 2022**
- **New additional node with 50TB SSD, 2 x 25 Gbps Ethernet**
- Xrootd version 5.4.0
- Current setup: xrootd in centos7 image with systemd-nspawn on debian.
  Full server configuration:
  https://gitlab.physik.uni-muenchen.de/Nikolai.Hartmann/xcache-nspawn-lrz/
- XCache settings:

```
pfc.ram 14g
pfc.blocksize 128k
pfc.prefetch 0
```

Data volume and reuse

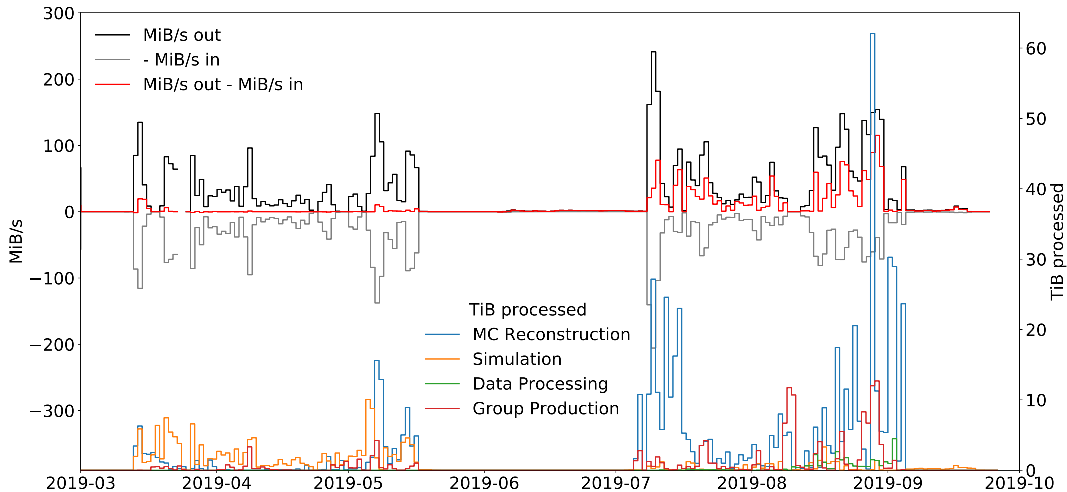


Caching most useful for workflows with frequently accessed data

Running in ATLAS production queue

Grid production

- First tests at the beginning of the project
  $\rightarrow$ reading data from closeby storage at MPP
  $\rightarrow$ also meant jobs scheduled for datasets available on MPP storage
- XCache worked well, single node could hold up with requests
  $\rightarrow$ few periods of main storage downtime where everything went through XCache
- Most data reuse came from pileup sample
- No streaming data - all data is copied to scratch disk



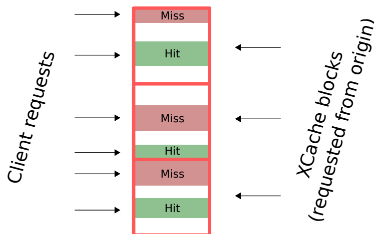
$\rightarrow$ largest hit rate for MC Reconstruction (here mainly pileup overlay)

Running in ATLAS "VP" analysis queue

Grid User analysis

- Analysis job data expected to be accessed more frequently
- Data read in streaming mode for ATLAS analysis jobs
  → can save network traffic for partial access (e.g. only several branches of a ROOT file)
  → XCache block sizes need to be adjusted appropriately (small enough)
  → currently working with 128k
- Virtual placement (VP)
  - "virtual placement" assigns datasets/files to cache sites
  - jobs scheduled based on that
  - distribute to individual caches of a site based on filename hash
    → handled via rucio
    → no redirector nescessary
  - can increase hit rate if configured appropriately

Note: monitoring hit rates with xrootd (cinfo or gstream)



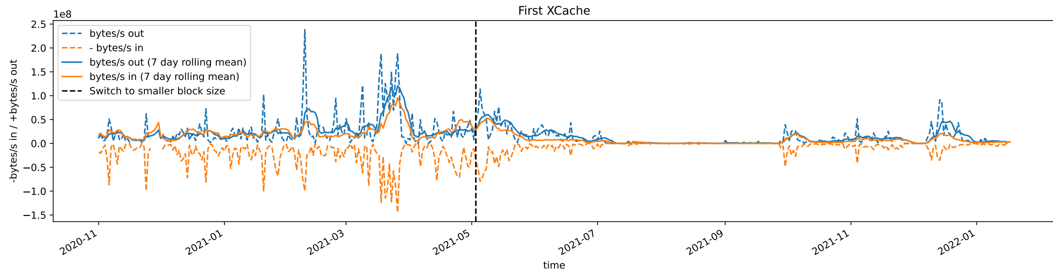
- Fetching of blocks and serving requests happens asychronously
  - might count as "served from cache" although download actually triggered by the same job
  - similar for requested chunks that span multiple (xcache) blocks or the other way round
  - happens also when downloading larger chunks (e.g. xrdcp) without vector reads because xrootd will split them up into smaller ones that are asychronously 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

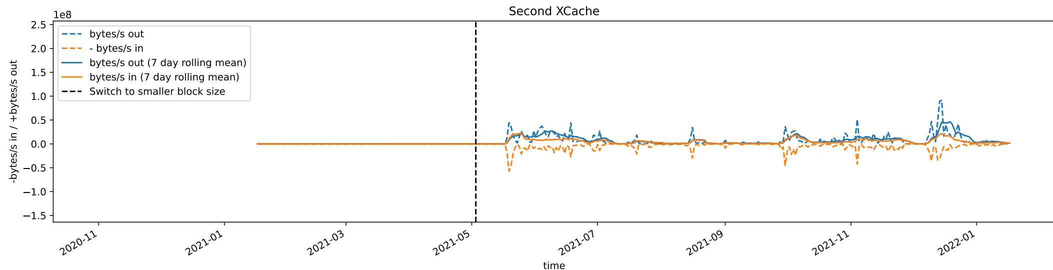
→ **"hit rate" calculated from this is not what we are mainly interested in ...**

→ switched to smaller block size (128k instead of 1M) - see if I/O defined hit rate improves

Did the switch to smaller blocksizes help?

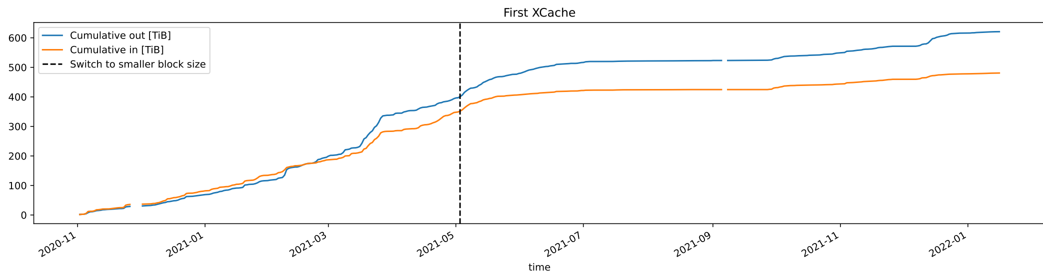
I/O rates:

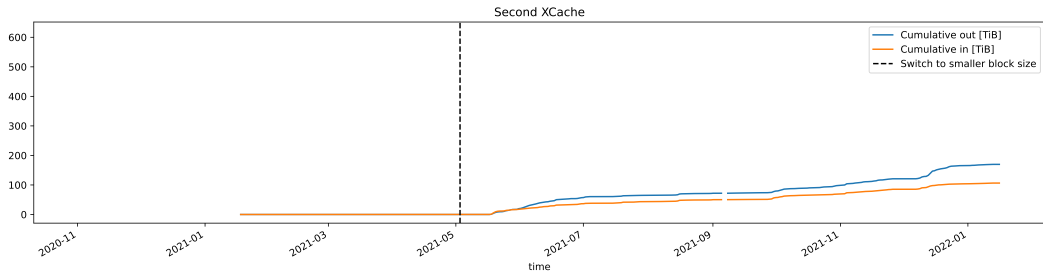




Did the switch to smaller blocksizes help?

Cumulative I/O:





Answer: Hit rate depends highly on type of workload
But: no longer periods of higher Input than Output observed after switch

XCache for I/O intensive last analysis step

Analysis Facility/Local processing

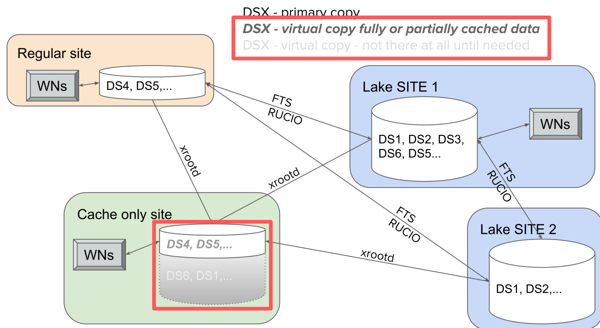
- End-user analysis often I/O limited
- Vectorized/Columnar data analysis increasingly popular
  $\rightarrow$ enables processing larger volumes of data in shorter time
- Interesting application for XCache
  - Some experiences from tests with ATLAS DAOD PHYSLITE
  - For spinning-disk based XCache the disks become a bottleneck
    $\rightarrow$ sparse reading for ROOT files with small baskets
  - SSD Cache expected to help here
  - Also optimizing data format (increase basket sizes, alternatives like parquet)
- Continuing to investigate this within FIDIUM

Summary, Conclusions, Recommendations

- Gathered a lot of of experience with XCache in different contexts
- Requirements change based on workflow, adjust setup/settings accordingly
  - Full file access? → prefetching useful
  - Streaming data? → no prefetching, smaller block sizes
  - I/O intensive analysis with scattered reads? → might need SSDs
  - Generally useful: Mount individul disks instead of RAID
- Data reuse typically increases going down the analysis chain
- Plans for nearterm future:
  - Continue running or XCache setup
    → also want to include other VOs, in particular Belle II
  - Continue testing virtual placement model at ATLAS
    → R&D, not required to do caching
    → still, potentially interesting model also for other experiments
  - Gather more experience for caching in last analysis step within FIDIUM

Backup

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 servers

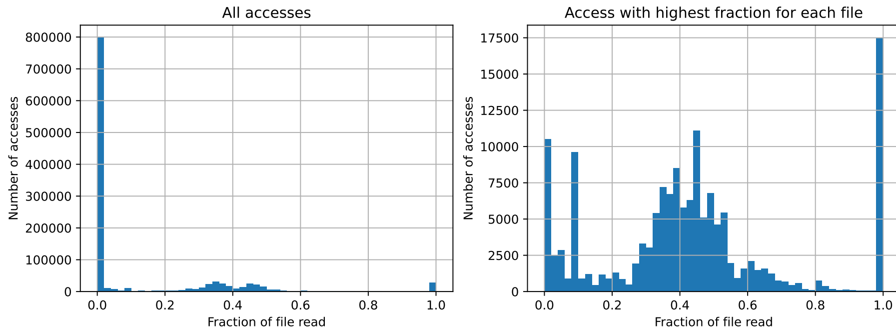
Monitoring of accesses in XCache

- Access statistics are tracked in `.cinfo` files (binary format)
- Monitored using g-stream monitoring in xrootd
- 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

Sparse reading of files

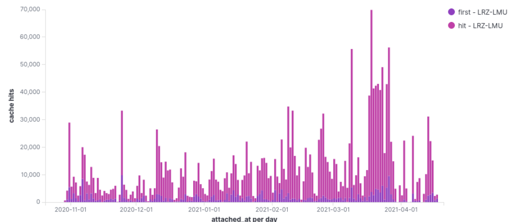
Analysis jobs typically access only a certain fraction of the data
(typically a fraction of all branches in a ROOT file)

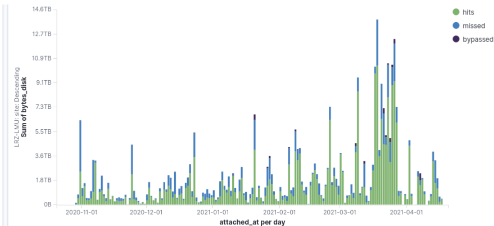
Data from last 2 months on our XCache analysis queue:

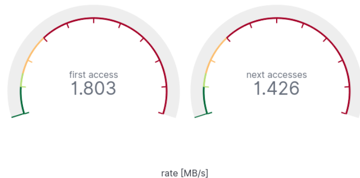


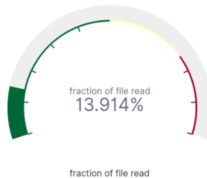
Hit rate from XCache monitoring

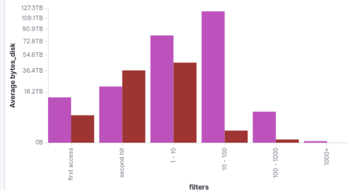
From ES Chicago database







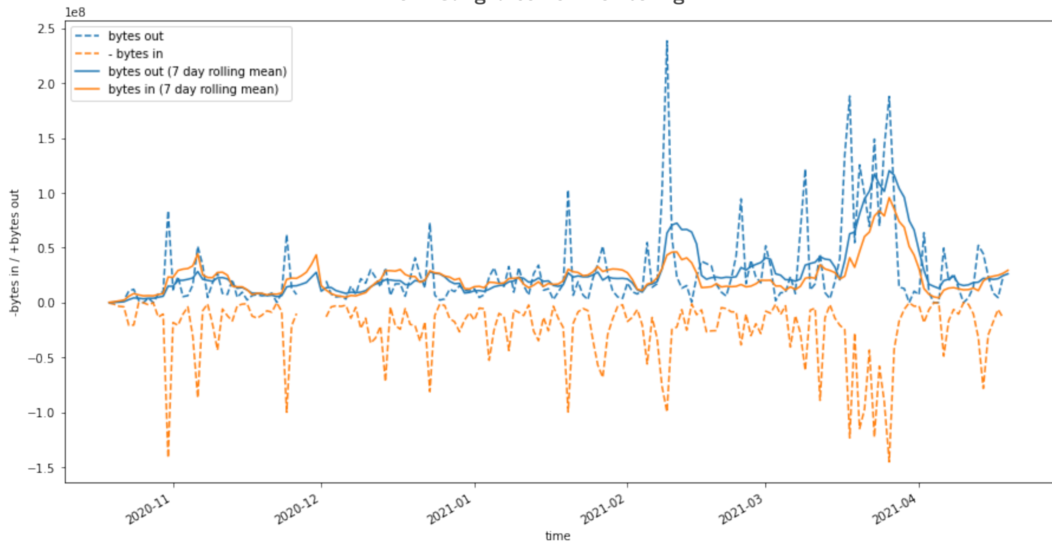




→ would expect much higher amount of output data than input data for cache

Hit rate in terms of data input/output

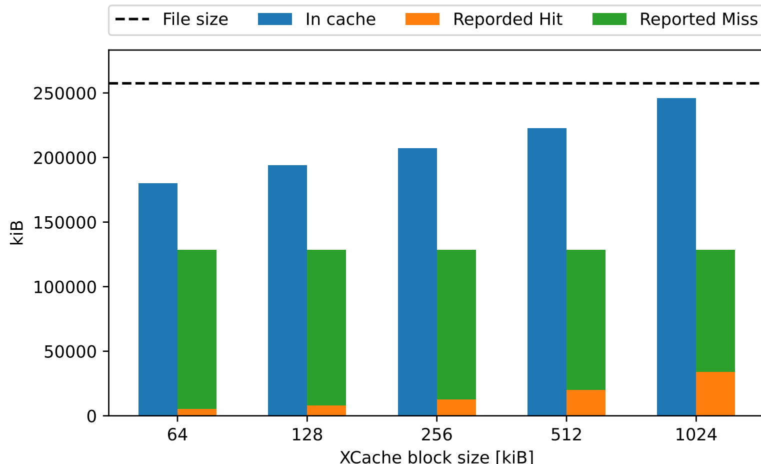
From Ganglia server monitoring



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

Try a smaller block size?

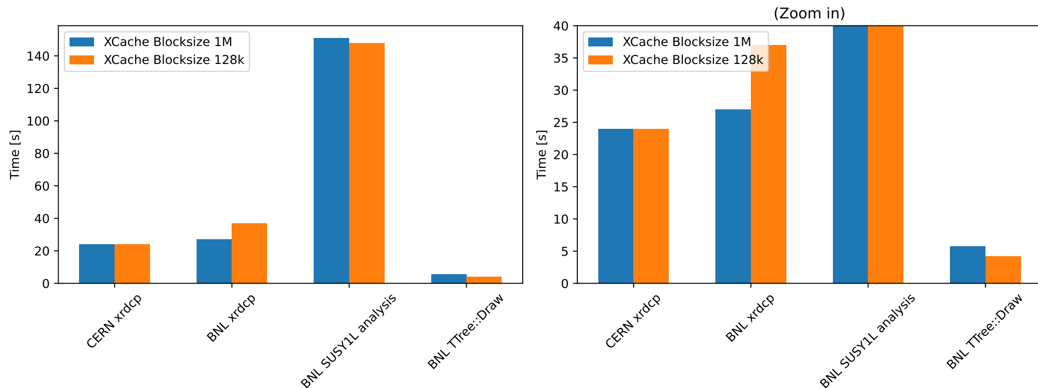
SUSY analysis example, running on a ttbar MC ROOT file, first access:



Previous setting: 1024 kiB
Currently testing: 128 kiB

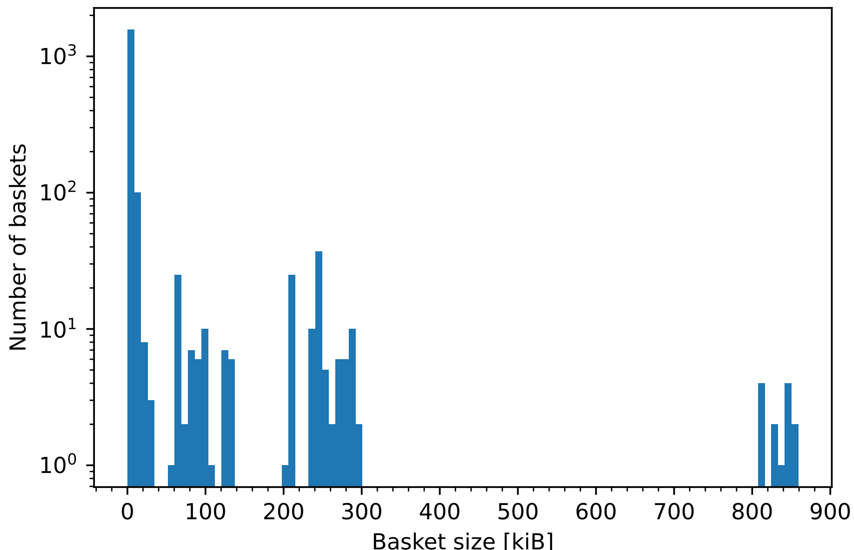
Impact of latency with smaller block size

XCache block size determines chunk sizes for reading from remote
$\rightarrow$ might impact performance due to latency (but chunks are requested async)



$\rightarrow$ some performance degradation for full file transfers with high latency (100ms)
$\rightarrow$ benefits from smaller amount of transferred data dominate for sparse access

Basket sizes (example for a ATLAS DAOD)



Columnar data analysis with ATLAS PHYSLITE

Idea:

- Most data is stored in "aux" branches (`vector<basic-c-type>`)
  $\rightarrow$ easily readable column-wise, also with uproot
- Reconstruction/Calibrations already applied
  $\rightarrow$ the rest might be "simple" enough to do with plain columnar operations
- Tools: uproot and awkward array

I/O ballpark estimate

Take current situation for ATLAS SUSY 1L analysis as an example:

- $\approx$ 200 TB of data with $\approx$ 50 kb/evt
- `DAOD_PHYSLITE`: $\approx$ 10 kb/evt
  $\rightarrow$ need to process $\approx$ 40 TB of data
  (probably a bit more since PHYSLITE unskimmed)
- Assume processing on local batch system: With 10 Gbit/s will take around 10 hours
  (with saturated network/150 kHz total)

Ways to improve this (get around network limit):

- Faster network connection
- Caching on the level of worker nodes
- "Intelligent data delivery" a la ServiceX
- Only read part of the data (few columns)
  $\rightarrow$ can be interesting e.g. for nominal only, early stages of an analysis
  $\rightarrow$ **study on the next slides**

I/O scaling tests

Some tests with a first PHYSLITE data sample (2015 data)

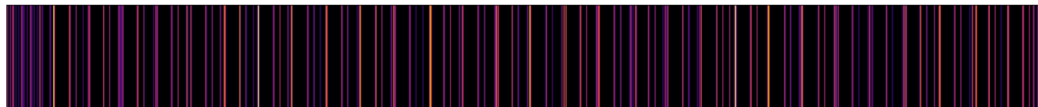
- read only few branches (same as in the 1L analysis test) of the $\approx$ 1 TB dataset
  $\rightarrow$ in theory $\approx$ **2% of the data**
- Read with $\approx$ **100 parallel tasks** (LMU batch system with dask)
- First test: read from LMU/LRZ dcache storage via xrootd (36 storage nodes)
  - Some issues with uproot (files not properly closed)
    $\rightarrow$ workaround, will be fixed in uproot4
    $\rightarrow$ some tuning of requested block sizes (will also not be nescessary anymore in uproot4, which supports xrootd vector reads)
  - After these fixes: data could be read within **5-10 minutes (7k files)**
  - Some storage nodes get rather busy with this access pattern
- Second test: read through xcache at LMU (1 storage node)
  - Gets extremely overloaded, not feasible anymore
  - Might become better when xcache is also extended to a cluster
  - But: maybe we can do better for this type of access (if we want to optimize for it)

Columnar data storage

- With current basket sizes in PHYSLITE these file accesses result in very scattered reading patterns
- A more columnar storage might help
- First try: store all “easily readable” branches in parquet files
  - Reading parquet via xrootd (using a small wrapper)
    (parquet files with 1 “row group” $\rightarrow$ one block per column)
  - Could read quickly ($\approx$ 3 min) even with single xcache node
  - Also good for block-wise caching (currently set to 1MB block size)
  - Not 100% fair comparison since only around 1/4 of data written to parquet
- Need to compare to very large basket sizes in ROOT
  $\rightarrow$ expect similar performance, but for the first test it was easier to produce the parquet files (awkward supports writing to arrow buffers)

Access patterns

Default DAOD_PHYSLITE



DAOD_PHYSLITE converted to parquet files (just “easily readable” branches)

DAOD_PHYSLITE with “jumbo baskets” (1 basket per branch)

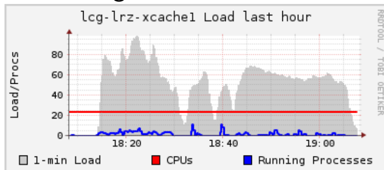
(plotting details: histogram (128kiB bin width) of number of bytes read, clipped at a maximum of 1000)

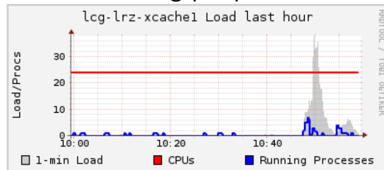
Monitoring plots

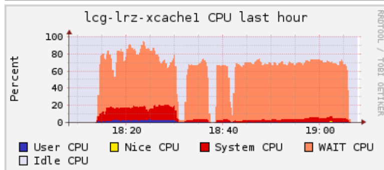
Reading default ROOT files

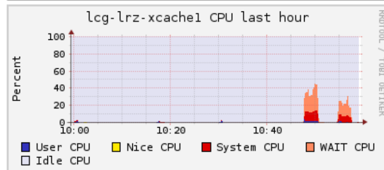
Reading parquet files

Load









I/O