XROOTD tests

Outline
- Castor background & changes
- What changes for you?
- XROOTD speed tests

Thanks to Andreas Peters!

Max Baak & Matthias Schott
ADP meeting
18 Jan '09

Castor

- CASTOR default pool: place to copy data to with commands like:
  - rfcp MyFileName /castor/cern.ch/user/<letter>/<UID>/MyDirectory/MyFileName
- Default CASTOR pool: 60 TB disk pool with a tape back-end.
  - When disk is full, older files are migrated to tape automatically to make space for newer files to arrive on disk.
- CERN has only one tape system, managed by Central Computing Operations group.
  - Shared by all experiments.

Castor problems

- CASTOR:
  - In theory:
    - "pool with infinite space". Sometimes delays to get files back from tape.
  - In practice:
    - Source of many problems and user frustration

- CASTOR problems:
  - Tape systems not designed for small files ➔ very inefficient.
    - Preferred file size >= 1Gb
  - If CERN tape system used heavily, long time for the data to be migrated back to disk ➔ applications time out.

- Typical user: uncontrolled/chaotic access to tape system
  - 'Lock up' when too many open network connections.
  - Could easily lead to situations which endanger data taking
    - Already several of such situations even without LHC running.

Changes to Castor

- LHC data taking mode:
  - Protect tape system from users to make sure it performs well/controlled when taking data.
- Consequence:
  - CASTOR pool becomes disk pool only
  - Default CASTOR tape back-end will be closed down for users
    - Disable write access for users to tape.
- Full information:
  - https://twiki.cern.ch/twiki/bin/view/Atlas/CastorDefaultPoolRestrictions

What changes for you?

- Files on CASTOR which have not been used for a long time, and which have been migrated to tape, will no longer be available.

- The CASTOR pool will be closed for writing in ~two months time
  - Original deadline January 15th , already extended!

- All data that is on the CASTOR pool that you want to keep need to be copied somewhere else.

- Copy whereto? Answer: “atlascernuserdisk” disk pool
- CERN personnel only, no disk quota
  - 100 Tb user disk. Average: ~1Tb / user
  - Not accessible through grid.
- Request permission: atlas-castordefaultpoolrestrictions@cern.ch

Instructions atlascernuserdisk

- atlascernuserdisk pool runs under a different CASTOR stager:
  - export STAGE_HOST=castoratlast3
  - export STAGE_SVCCLASS=atlascernuserdisk

- Then simply use the usual rfcp, rfdir, ... commands.
- Copy to usual $CASTOR_HOME directory
  - /castor/cern.ch/user/<letter>/<UID>/

- Transition period to bring data into this pool from the default tape system:
  - (slow)
  - export STAGE_HOST=castoratlast3
  - stager_get -M $CASTOR_HOME/<subdir>/<filename> -S atlascernuserdisk

- Request permission: atlas-castordefaultpoolrestrictions@cern.ch

XROOTD

- For default CASTOR pool, files can be accessed directly in root / athena using RFIO protocol
  - Filenames begin with “rfio:” turl.
  - Eg. rfio:/castor/cern.ch/user/m/mbaak/dummyFile.root

- atlascernuserdisk disk pool connected to XROOTD server

- XROOTD: file server/network protocol developed for root
  - Easily scalable to larger file systems, handle many open network connections
  - Well integrated in PROOF (parallel processing)
  - Originally developed at SLAC
    - Used successfully in BaBar experiment
    - “Fast and reliable”

Requirements for using XROOTD

- Root v5.18e or greater
  - Athena 14.5.0 or greater
- Athena: need to create a PoolFileCatalog.xml file

- xrootd privileges
  - atlas-castordefaultpoolrestrictions@cern.ch
- Environment variables
  - export STAGE_HOST=castoratlast3
  - export STAGE_SVCCLASS=atlascernuserdisk

- The files copied to "atlascernuserdisk" can be accessed in root or athena via:
  - File prefix: root://castoratlas3/
  - root://castoratlast3//castor/cern.ch/user/m/mbaak/dummyFile.root
    - TFile* foo = TFile::Open("root://castoratlas3/file_on_atlascernuserdisk");

XROOTD test results

Outline
- Single job performance
- Stress test results
  - (Multiple simultaneous jobs)

Test setup

Five file-transfer configurations:

- Local disk (no file transfer)
- FileStager
  - Effectively: running over files from local disk
- Xrootd
  - Buffered
  - Non-buffered
- Rfio

Files used:

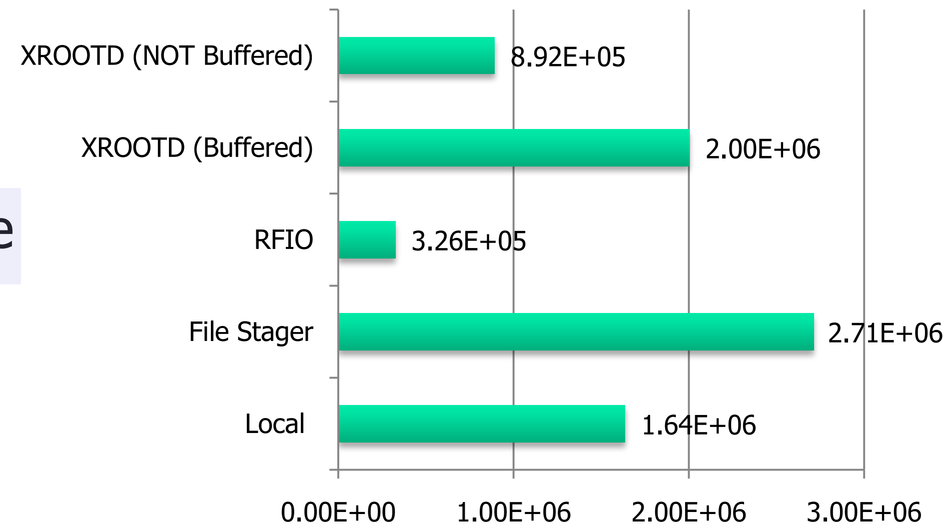
- Z➔ee, Z➔mumu AOD collections
- ~37 mb/file, 190 kb/event
- 200 events per File

Intelligent FileStager

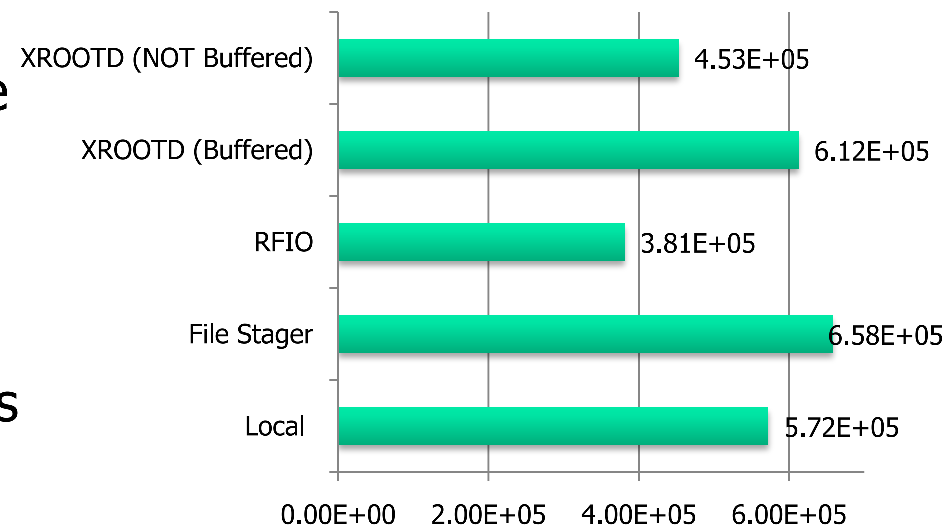
- cmt co –r FileStager-00-00-19 Database/FileStager
  - https://twiki.cern.ch/twiki/bin/view/Main/FileStager
- Intelligent file stager copies files one-by-one to local disk, while running over previous file(s).
  - Run semi-interactive analysis over files nearby, eg. on Castor.
  - File pre-staging to improve wall-time performance.
  - Works in ROOT and in Athena.
- Actual processing over local files in cache = fast!
  - Only time loss due to staging first file.
  - In many cases: prestaging as fast as running over local files!
  - Minimum number of network connections kept open.
  - Spreads the network load of accessing data over length of job.

Large Scale Tests

- Basic Test Setup
  - Number of Files: 259
  - Total Datavolume: 9.55 GB
  - Number of Events: 52.000
  - Only one job is execute on a batch machine



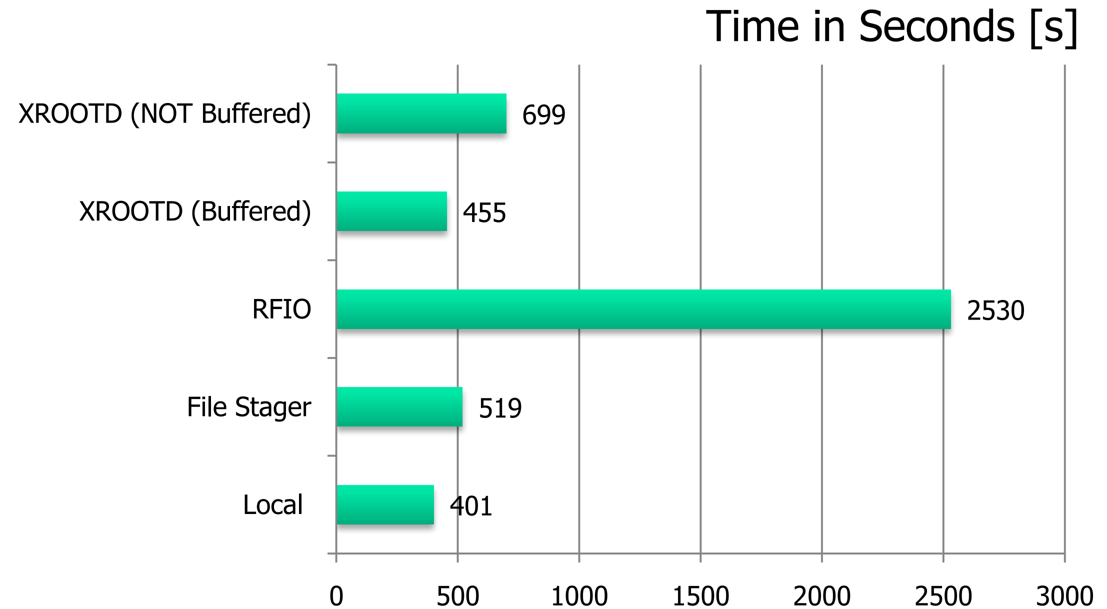
- Three different Tests
  - Setup 1: Read only Event Number
  - Setup 2: Read 7 containers
  - Setup 3: Read 7 containers + Monte Carlo Truth Information + some Algorithmic
  - Setup 4: Same as Setup 2 but with Tag-File Access

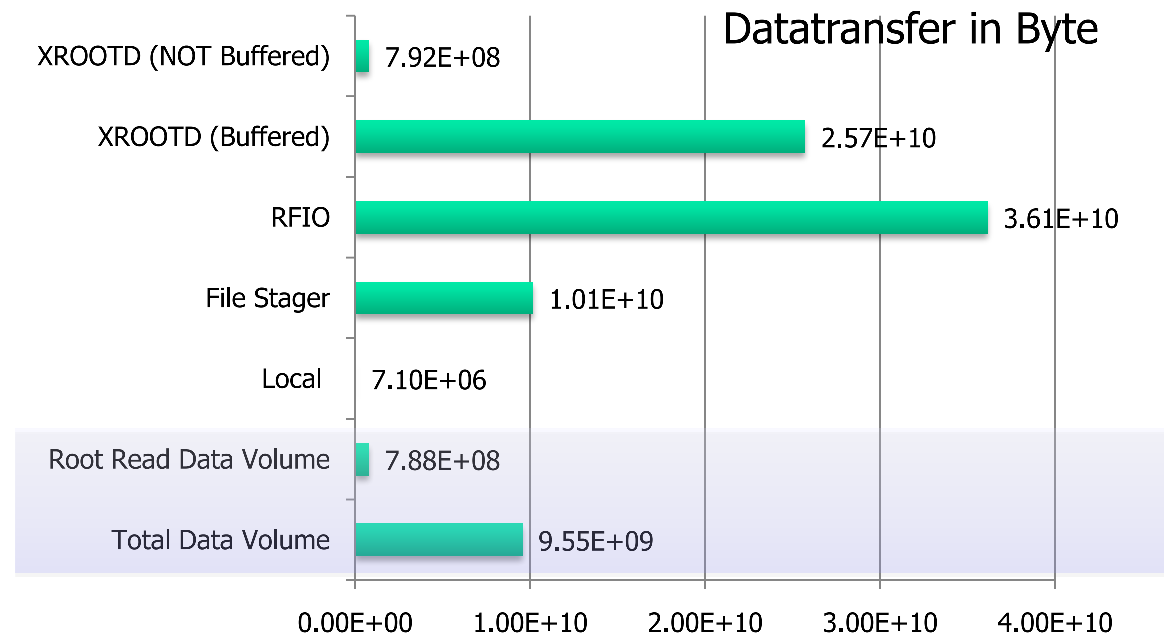


- Note
  - In this test the file access is overstressed as data-AOD files have a supposed file-size of 2GB and contain much more events

Large Scale Tests

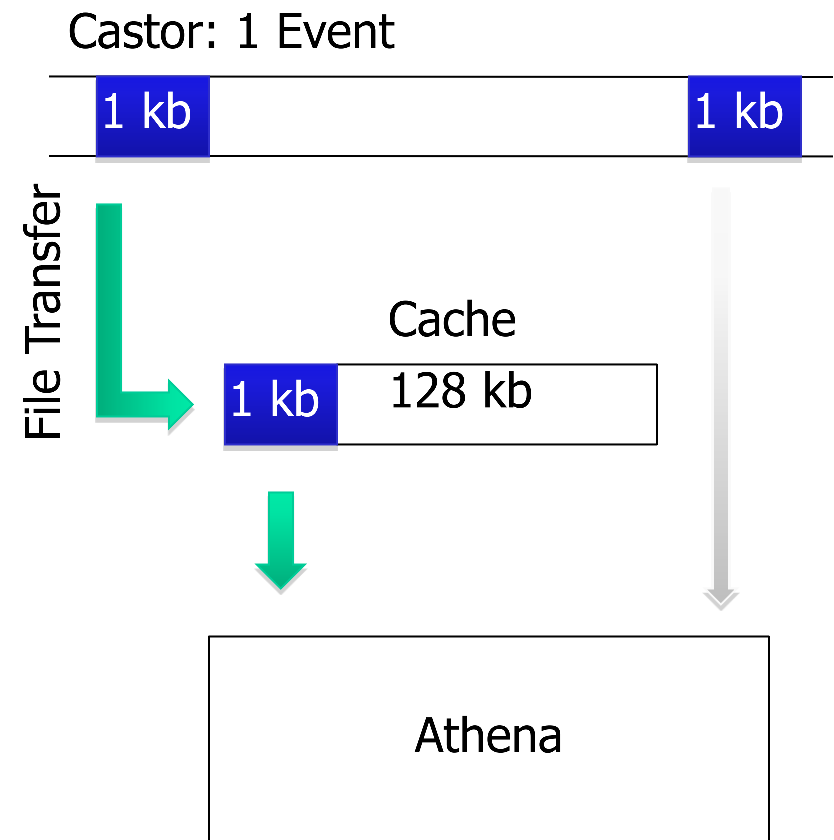
- **Reading Only Event Number**
  - Reading 10% of File Content
- **Timing:**
  - Comparable timing for local access, xrootd and file stager
  - RFIO 5 times slower
- **Datatransfer**
  - **RFIO:** 45x larger data transfer than needed
  - **xrootd (not buffered):** 1% overhead of file transfer
  - **xrootd (buffered):** 32x larger data transfer than needed
  - **FileStager:** Copies whole file and hence 12x larger data transfer than needed





Read-ahead buffer

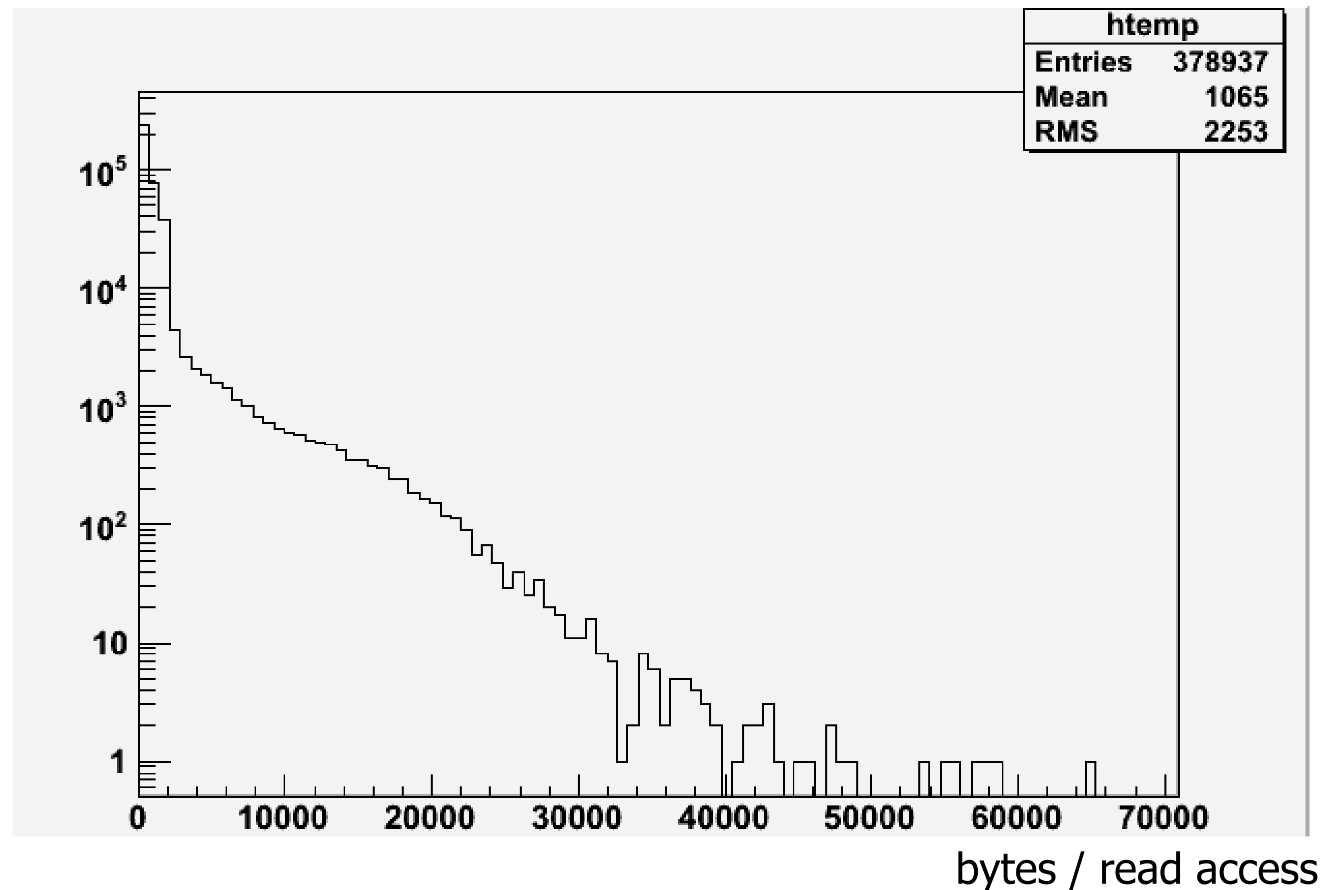
- After read request, read ahead YYY kb
  - In anticipation of next read request.
- Read-ahead buffer transferred and stored in cache.



- Xrootd:
  - Read-ahead: 512 kb
  - Cache size = 10 mb
- Rfio:
  - Read-ahead: 128 kb
  - Cache size = read-ahead size (128 kb)
    - Effectively not used.
- Unfortunately: caching not very successful for our purposes

- Xrootd: Read-ahead buffer can be turned off.
- Rfio: can probably be turned off as well, but we didn't manage.

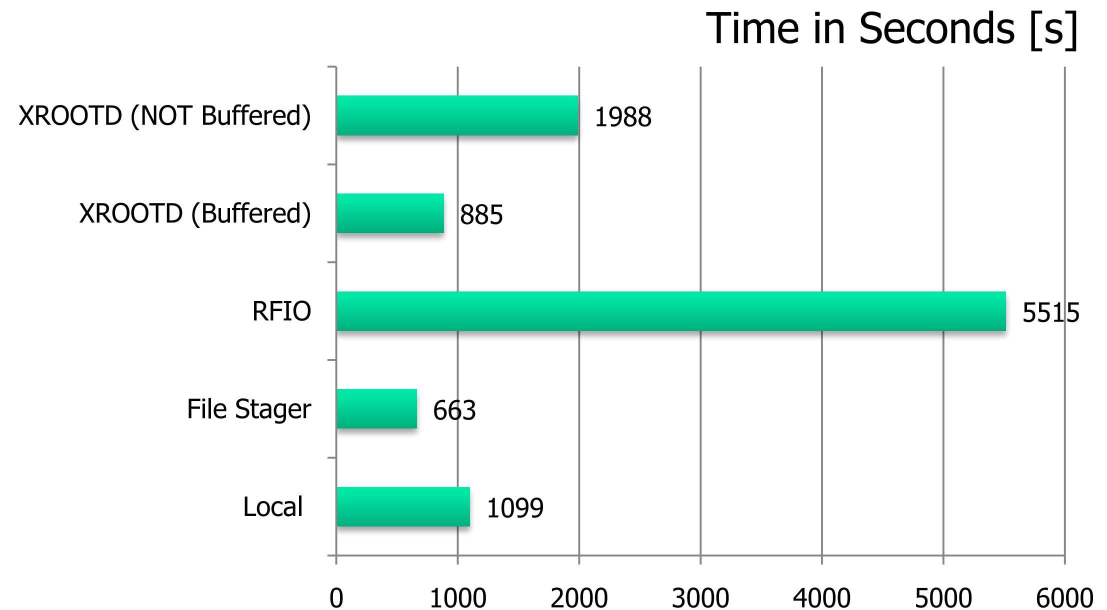
Typical read access pattern

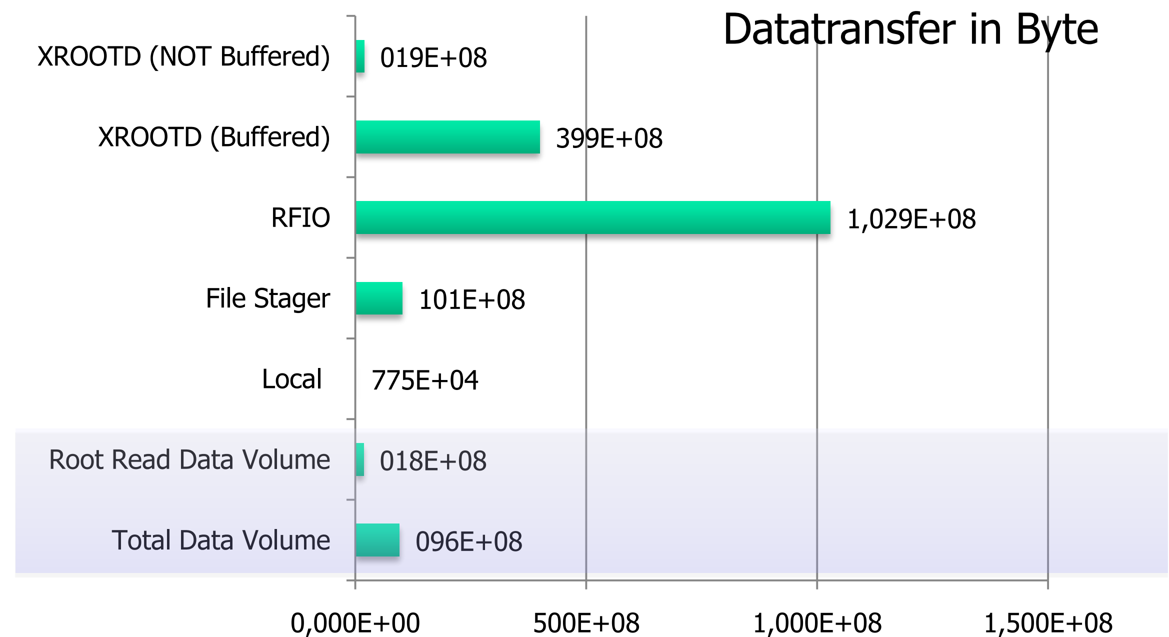


- Typical AOD read access pattern.
- Average: ~1 kb / read access
- Note: 128 kb read-ahead buffer

Large Scale Tests

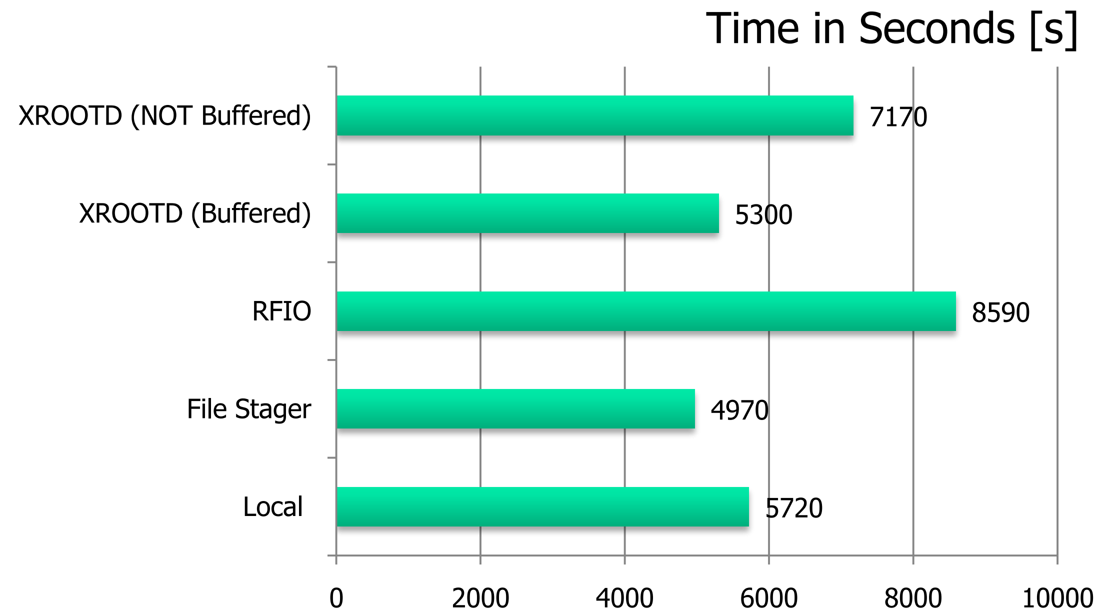
- **Analysis I:**
  - Reading 20% of File Content
- **Timing:**
  - RFIO 5 times slower
  - Xrootd (not buffered) twice as slow as local access
  - File Stager faster than local access, as files are still in cache when loaded by Athena
- **Datatransfer**
  - **RFIO:** 57x larger data transfer than needed
  - **xrootd (not buffered):** 5% overhead of file transfer
  - **xrootd (buffered):** 22x larger data transfer than needed
  - **FileStager:** 5.5x larger data transfer than needed

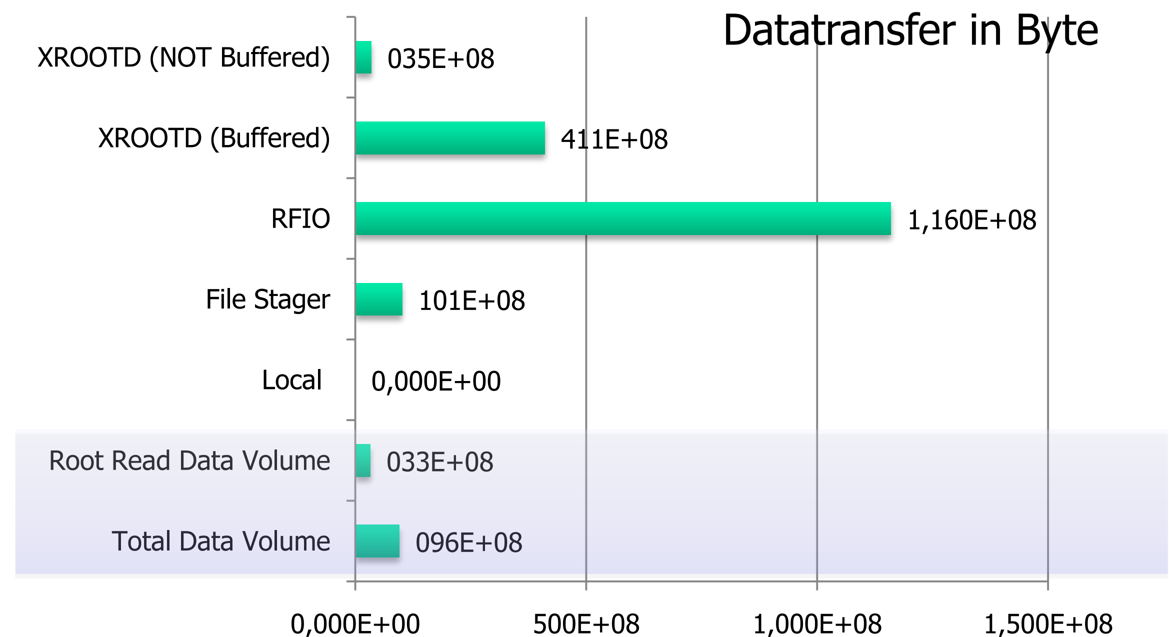




Large Scale Tests

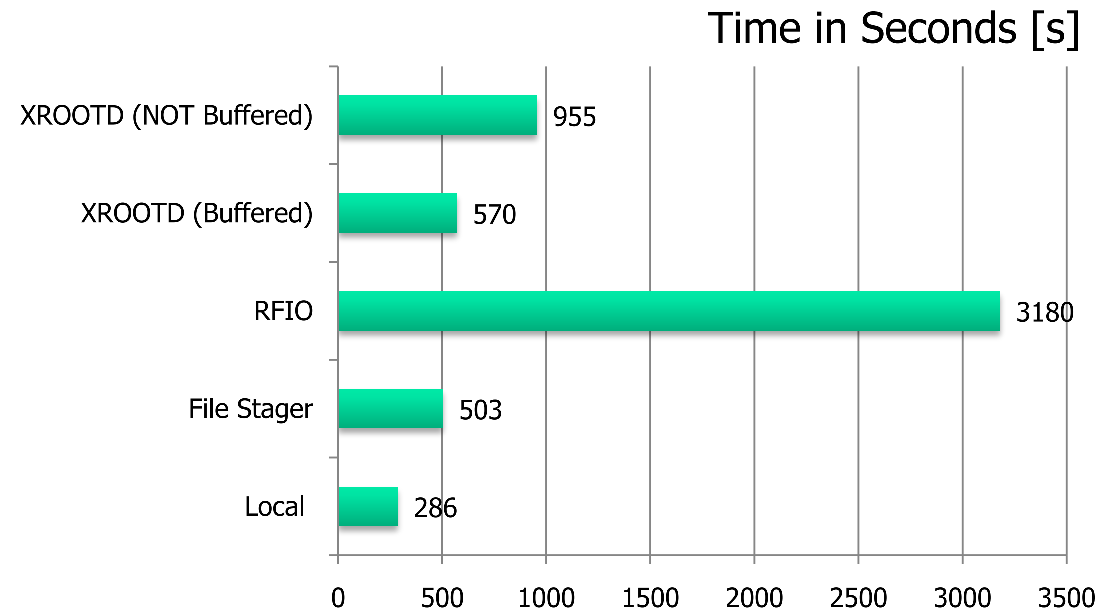
- **Analysis II:**
  - Reading 35% of File Content and more algorithmic inside the analysis
- **Timing:**
  - Overall comparable timing as algorithmic part gets dominant
  - Xrootd (not buffered) is 20% faster than RFIO.
  - File Stager faster than local access, as files are still in cache when loaded by Athena
- **Datatransfer**
  - Similar to previous analysis

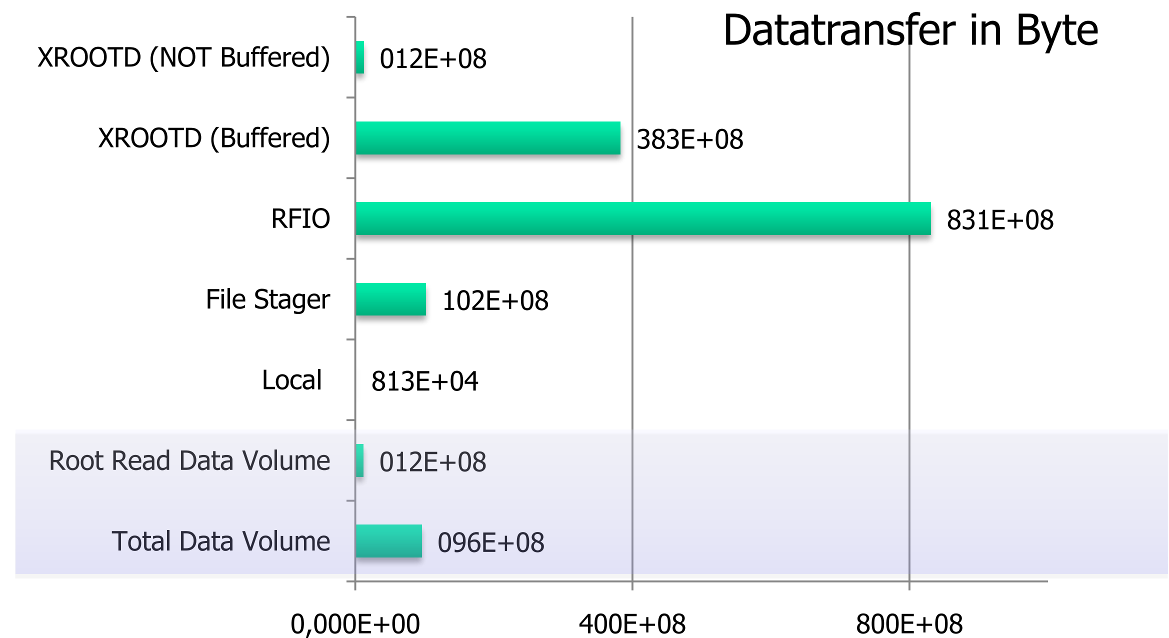




Large Scale Tests

- **Analysis I with Tag-Files:**
  - Reading 20% of File Content
  - Access only 20% of the events in each file
- **Timing:**
  - Local processing is the fastest (2x faster than the next)
  - RFIO is dominated by latency of opening the files
  - Xrootd 3x faster then rfio:
- **Datatransfer**
  - Similar to previous analysis





Stress- & Timing test

- Stress test: many similar jobs running simultaneously
  - Focus of timing
- Time measured = time looping over events. No initialization.

- Protocols tested:
  - FileStager
  - Xrootd (no buffer)
  - Xrootd (w/ buffer)
  - rfio

- Basic Test Setup
  - Number of Z colls: 20
  - Total Datavolume: 700 MB
  - Number of Events: 4000
  - Setup 2: Read 7 containers
    - 20% of file contents

Xrootd: PoolFileCatalog

- Athena needs PoolFileCatalog.xml file to run over collections using xrootd.

- Use command: pool_insertFileToCatalog
  - Very unstable and slow!

- Cmd has hard time handle more then ~100 files per catalog
- For 20 files/catalog, (random) crash rate of ~40%.
- Slow: 100 files takes 20 sec

- Numbers on xrootd presented next are slightly optimistic.
  - Many xrootd jobs crashed, freeing up bandwidth/cpu for the stress-test.

- pool_insertFileToCatalog command needs urgent fixing.
- Best solution: no dependency on PoolFileCatalog.xml
  - Like for local files or rfio.

One vs Many jobs

- Running one job

Protocol	Njobs	Avg. time (s)	Factor	
FileStager	1	66.7	1.00	
Xrootd (no buf)	1	109.4	1.64	
Xrootd (w/ buf)	1	101.0	1.51	
rfio	1	289.8	4.34	

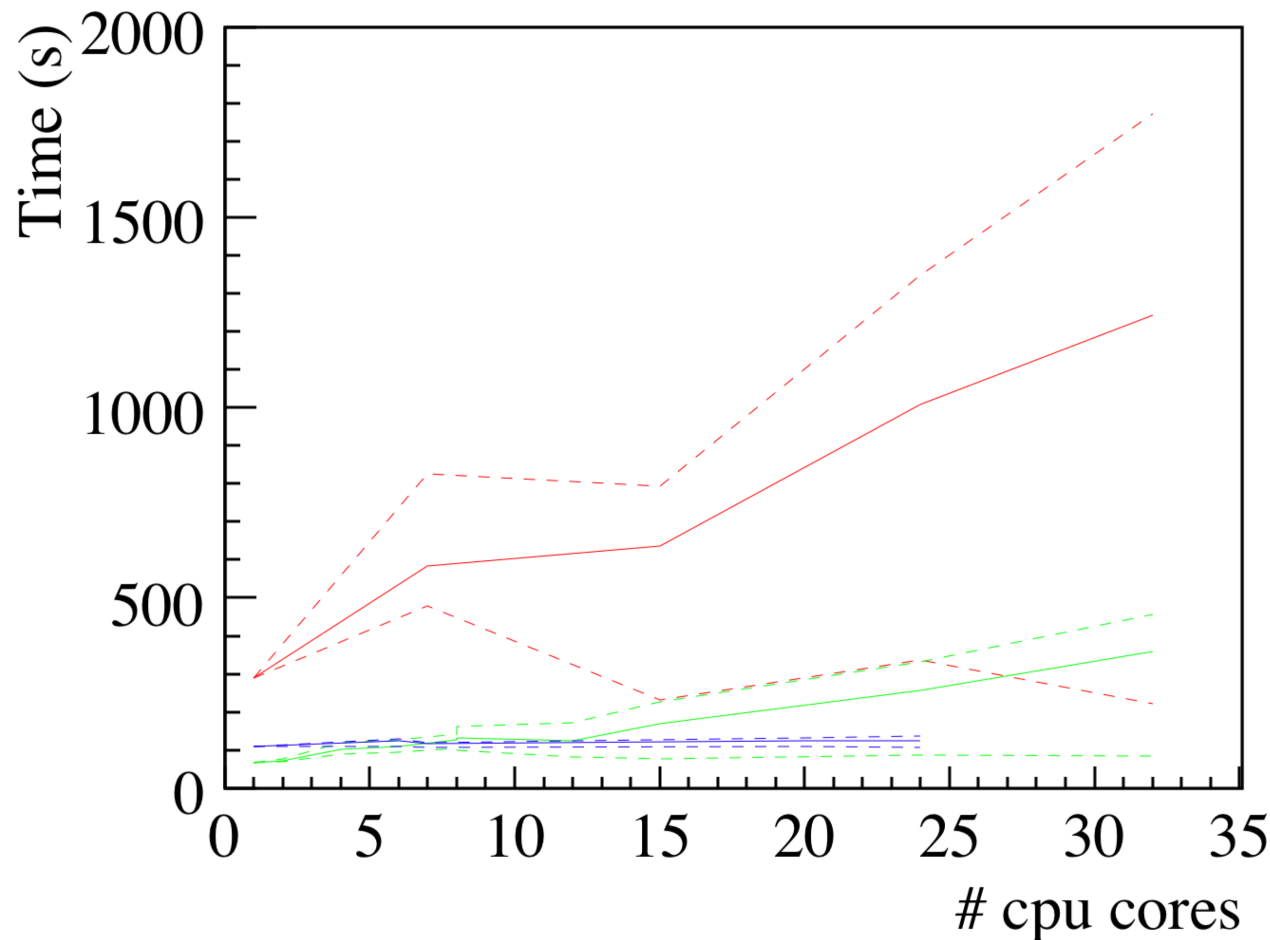
- Many jobs running simultaneously, over different, (mostly) *uncached* collections

Protocol	Njobs	Avg. time (s)	Min (s)	Max (s)
FileStager	32 (32 cores)	262.1	207.7	381.5
Xrootd (no buf)	15 (32 cores)	160.9	119.5	228.5
Xrootd (w/ buf)	16 (32 cores)	234.5	78.9	723.7
rfio	24 (32 cores)	627.8	307.1	928.6

- 4 nodes, dual quad-core. Uniform filling of cpu slots.
- Lxbatch node: 1Gbit ethernet card / node
- One buffered job can easily clog up entire network bandwidth!

(Cached) Timing tests

- Test: identical jobs running over identical cached collections



Average of:

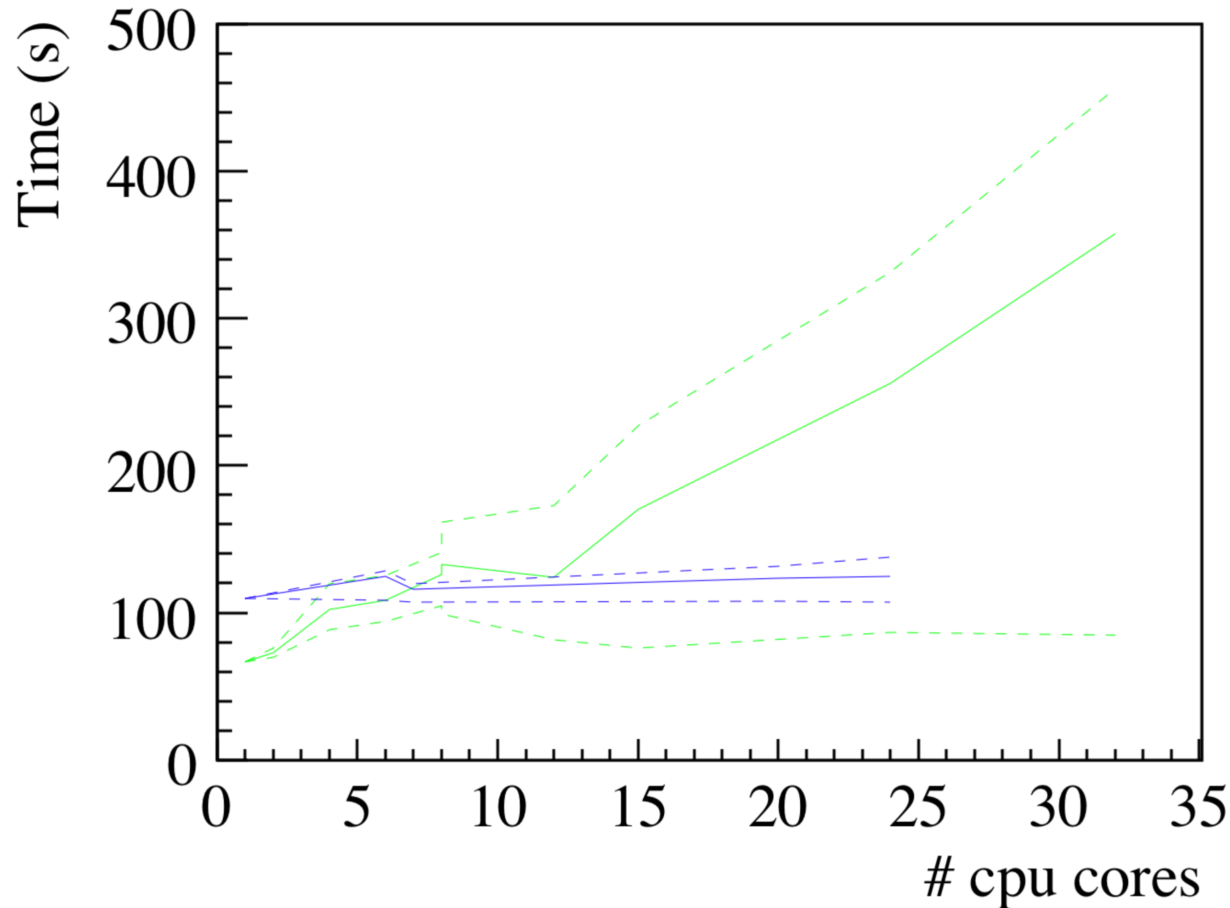
- Rfio
- Xrootd (no buffer)
- FileStager

- Dashed lines: min/max values of test

- Rfio: terribly inefficient.

(Cached) Timing tests

- Test: identical jobs running over identical (cached) collections



Average of:

- Xrootd (no buffer)
- FileStager

- Dashed lines: min/max values of test

- Each job runs over same collections.
- Timing of xrootd is very stable running over cached files!

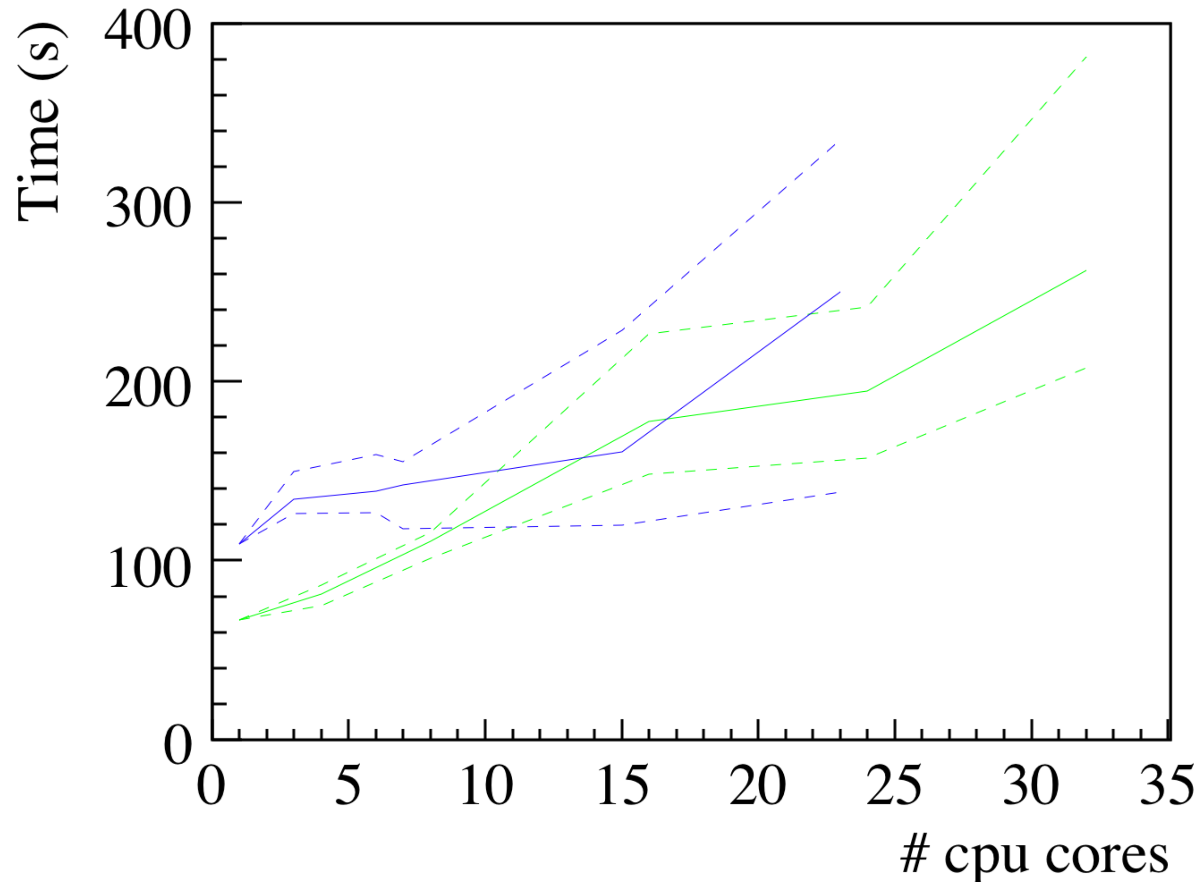
Stress test

- Many jobs running simultaneously, over different cached collections
- Strategy: flood the batch queue.

Protocol	Njobs	Avg. time (s)	Min (s)	Max (s)
FileStager	189	582.5	68.9	1004.8
Xrootd (no buf)	111	169.7	124.7	239.8

(Uncached) Timing tests

- Test: similar jobs running over different, uncached collections



Average of:

- Xrootd (no buffer)
- FileStager

- Dashed lines: min/max values of test

- 4 nodes, dual quad-core. Uniform filling of cpu slots.
- X-axis: n jobs running simultaneously. Y-axis: time / job
- Each job runs over same collections.

Preliminary recommendations

- Xrootd & rfio read-ahead buffering: very inefficient
  - Lots of unnecessary data transfer (sometimes >50x data processed!)
  - 1 job completely blocks up 1Gbit ethernet card of lxbatch machines
    - ➢ Large spread in job times, ie. unreliable

- Xrootd: frustrating dependency on PoolFileCatalog.xml

- Don't use rfio protocol to loop over files on CASTOR!
  - >5x slower & takes up too much network bandwidth

Preliminary recommendations

Different recommendations for single / multiple jobs

- Single jobs: FileStager does very well.
- Multiple, production-style jobs
  - Xrootd (no buffer) works extremely stable & fast on files in disk pool cache.
    - Factor ~2 slow-down when read-ahead buffer turned on.
  - Two recommendations:
    - Xrootd, no buffer, for *cached* files
    - FileStager or Xrootd for *uncached* files