Storage performance issues

Charles G Waldman
University of Chicago

USATLAS Facility Workshop, Oct 12-13 2010

Storage implementation choices

- Storage on worker nodes, or dedicated storage hardware?
  - T1 and T2 started with the “shared” model but have mostly moved to dedicated
    - Typical: Dell 2950 (8-core) host with 8 MD1000 disk shelves (~20TB/shelf)
  - T3s will probably use shared model, with xrootd
- How much to take advantage of non-local data? (storage federation)

Stage-in, without caching

- Files copied to WN /scratch and cleaned up after job completion (no reuse)
- Files may be read multiple times, for checksum calculation after transfer
- Entire file read, even if only a fraction needed
- Creates lots of local disk I/O
  - this gets worse as #cores/#spindles increases
- Some robustness against svc. interruptions

Stage-in, with pcache

- lsm-get or other wrapper intercepts copy requests
- Files copied to /scratch/pcache/... and hard-linked to work dir
- Cache managed automatically (LRU)
- Callbacks to Panda server for file locality (GUID/LFN)
- Reduces copy-in by up to 50%

Direct-access

- Lustre/Hadoop/Xroot/...
- Xrootd: known to work well @SLAC
- Proposed solution for T3's
  - Will servers be capable of handling shared storage/compute load? Can node-aware job brokering help?
- Downsides: less data-integrity checking, possibly more brittle

Direct-access 2

- Good for jobs with sparse data access patterns, or files which are not expected to be reused (analysis jobs, skims)
- Currently testing at MWT2/AGLT2 (dCache)
- Same amount of data (or less!) moved, but latency is a consideration since job is waiting

Node-level job brokering (for T3s)

Proposal: reuse mechanism implemented by Tadashi for pcache

Memcache-based DB on Panda server tracks files on WNs, via http POST requests:

- addFilesToCacheDB
- removeFilesFromCacheDB
- checkFilesWithCacheDB
- flushCacheDB

Node-level job brokering, 2

Need to integrate this with xrootd

- Possibility of using 'inotify'- or 'dnotify-' based daemon to handle HTTP callbacks (inotify is fairly new,not supported in all kernels)
- Or else, some development with XRD required...

dcap direct-access test at MWT2

- Many jobs hanging when direct-access was first enabled...
- dcap direct access is a less-tested code path
- Invalid inputs causing hangups due to brittleness in dcap protocol (buffer overflows, unintentional \n in file name)
- All job failures turned out to be due to such issues (sframe, prun...)
- dcap library patch submitted to dcache.org

dCache tuning

- Movers must not queue at pools!
  - set max_active_movers to 1000
- Setting correct ioscheduler is crucial
  - cfq = total meltdown (throughput, not fairness!)
  - noop is best – let RAID controller handle it
- Hot pools must be avoided
  - spread datasets on arrival (space cost=0), and/or use p2p. “Manual” spreading so far not needed
  - HOTDISK files are replicated to multiple servers

dcap++ (LCB: Local Cache Buffer)

- Gunter Duckeck, Munich (link)
- 100 RAM buffers, 500 KB each
  - Hardcoded, needs to be tuneable
  - Sensitive to layout of ATLAS data files
  - Tuned for earlier release, 500KB is too big
- In use in .de cloud (and mwt2) w/ good results
- Awaiting upstream merge (6 months pending)

Read-ahead in general

- Needs to be flexible so parameters can be tuned for different ATLAS releases or user jobs (advanced user may want to control these values themselves)
- No “one-size-fits-all” answer

Some results

- CPU/Walltime efficiency (rough #'s):

	Local I/O	Remote I/O
stage-in	~40%	~40%
dcap	65%	~35%
dcap++	78%	~55%
xroot	78%	40%

Recommendations

- Direct-access does not handle all file types, only ROOT files … other files are still staged.
- DBRelease files are most likely to be reused, hence, pcache is still applicable, even with direct-access.
- Use stage-in/pcache for production, direct-access for analysis.
- Pursue enhanced caching for xrootd remote I/O.
- Encourage storage federation.

References

stage-in vs direct-access studies

pcache notes (pdf)