CMSSW I/O Update

‘The Big Picture’

- Just like last update, we’re fairly happy with CPU efficiency.
  - The majority grid use is ntuple-creation.
  - Many users are re-constructing a subset of the event or doing other CPU-intensive work.
    - The I/O-intensive stuff exists, just not yet at an appreciable scale on the grid.

User studies

- With the version of CMSSW for the 2012 run (CMSSW_5_2_x), we're just now starting to look what users will do.
  - Of the 3-4 analyses we've looked at, the "most interesting I/O" one involves reading about 1/3 of the data in the AOD.
  - About 10 complex, top-level object collectionss. Broken up into about 500 branches
  - The full collection is read for each event. However, only a part of the collection is used.
  - If there was a way to *safely* read part of the event, it would make a dramatic increase in the I/O rates.
    - We don't see anything that doesn't appear to be a loaded gun pointed at foot.

Current I/O Tests

- Doing some "touch-ups" now that we're on ROOT 5.32:
  - Making our statistics uniform by adding them for Xrootd protocol.
  - Less-aggressive read-ahead, but enabling vector-reads for RFIO & dCap.

Xrootd Changes

- We are working to have the xrootd client to be used through the CMS I/O "layer", not TXNetFile.
  - Allows us to do more thorough accounting statistics than TFile.
  - Allows us to use lazy-download (pulls complete file to /tmp) on "known bad" workflows: generators and merging.
    - Would really love to see merging fixed...

CMS Statistics

- Our newest computing system aggregates the per-job statistics into per-workflow numbers.
  - We're hoping this results in more "big picture" statistics.
  - Likely, only for "central tasks", not analysis this year.

Touch-ups

- We are resetting TTreeCache for backward jumps *if* the backward jumps are sufficiently large.
- Approximately 0.1% of our files are "pathological" in that the lumi section organization causes TTreeCache to basically turn off.
  - Translates to horrible performance when combined with read-ahead.

Touch-ups

- We do read-coalescing to decrease the number of separate reads -- at the cost of increased total data read.
  - The (old) naive implementation disables readv in order to avoid excessive buffering and copies.
  - The newer implementation uses the ROOT-provided buffer as scratch space, plus a fixed-size temporary buffer:
    - Allows us to use read-coalescing, vector reads, and a fixed amount of buffer space while only adding (on average) one round trip.

New Merge Algorithm

- Recently handed Philippe a new merge sorting algorithm that is “cluster aware”.
  - Make sure all baskets in a branch are grouped together within the cluster.
  - Make sure all child branches are grouped with their parents in the cluster.
  - Sort in descending order of total top-level branch byte size.
- Actually, fairly simple to implement in the end.

Async Prefetch

- As promised, we did some testing with async prefetch.
- It was about a 10% slowdown on the LAN, and a great improvement on the WAN.
  - The higher the latency, the greater the improvement.
  - At some point, the performance graphs change-over and the async prefetch is the best choice.
- Because it's not a clear win in all situations, it'll likely be disabled by default - and hence never used.
  - We'd *really* like the ROOT team to study optimizing this so we can safely always turn it on.

Work left undone

- We've taken a few honest stabs at implementing a "smarter" basket layout algorithm.
  - It's proven very hard to get right.
  - We've exhausted the clock on current attempts: likely won't be able to think about it again until the summer.

Threading!

- Since CMSSW6 is likely to be multi-threaded, thread-safety is the current hot topic in CMS.
  - ROOT I/O is very non-thread-safe, especially in the de-serialization layer.
  - Make it so? All things considered, this might be our "top ask" to the ROOT team.