Object Stores for CMS data

Nick Smith

HSF Analysis Ecosystems Workshop

24 May 2022

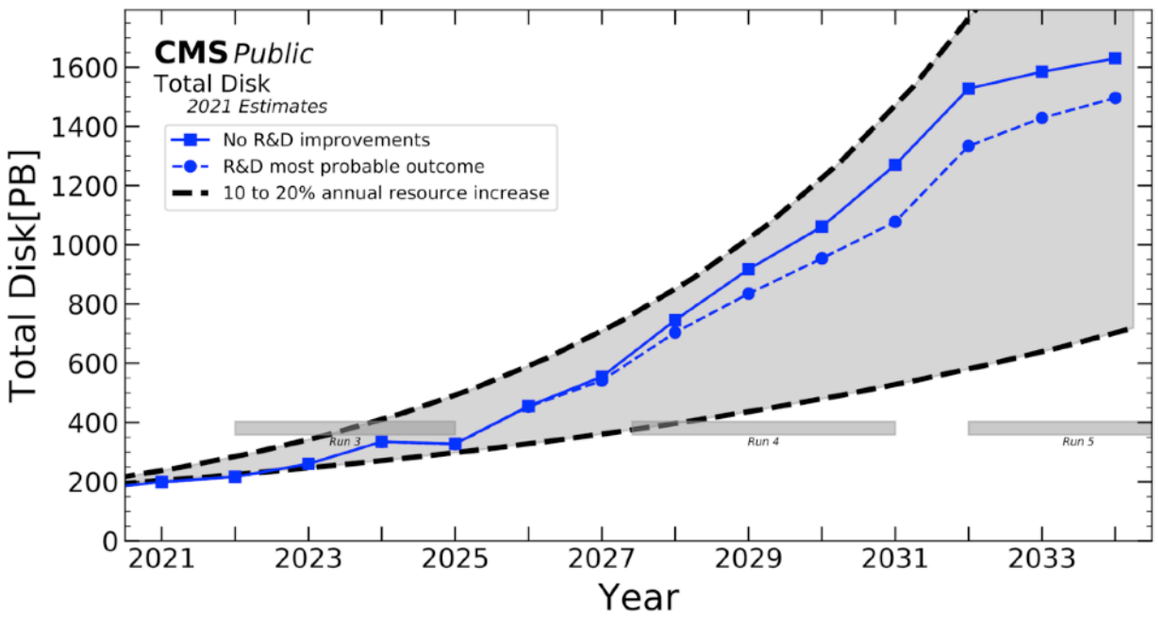
Rothko, Number 19 (1949)

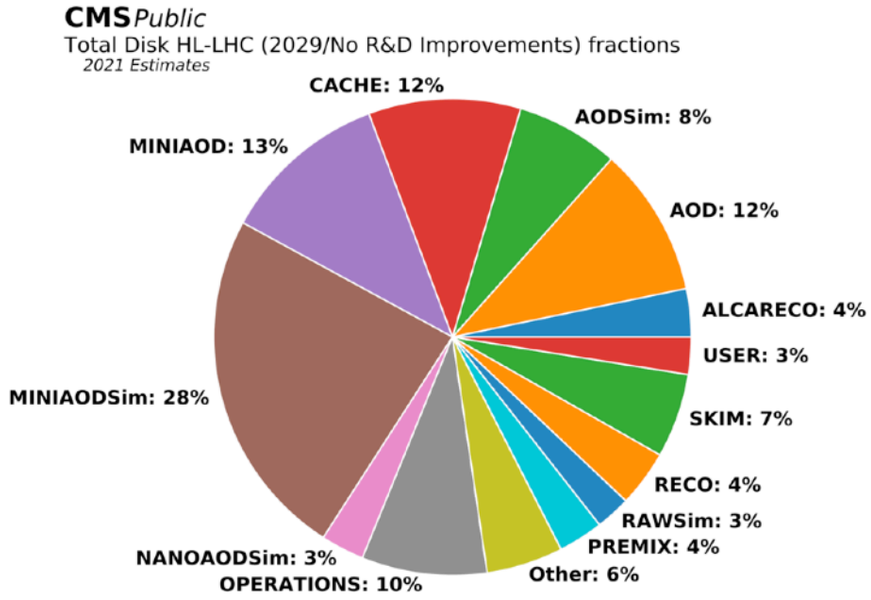
Intro

- CMS stores events (rows) in *datasets* (~100k) each containing many *files* (~100M) which have several *replicas* stored at sites around the world
- *Primary datasets* are an abstract concept “what kind of events”
  - A primary dataset (PD) will have many associated datasets containing different information about some set of events drawn IID from a distribution specific to the PD
- *Data tiers* statically define what subset of information (columns) are included in a given dataset
  - (GEN -> SIM -> DIGI ->) RAW -> RECO -> AOD -> MiniAOD -> NanoAOD
  - Progressively less bytes but more high-level information about the events
- Analysis uses the smallest practical data tier
  - Each analysis will need a different set of data columns
  - Changes to tier definition require an expensive reprocessing
  - We forward-copy many columns from one tier to the next (e.g. LHEEventProduct)

Disk is a cache

- Disk is expensive: offload as much as possible to tape
  - Only MiniAOD, NanoAOD tiers reliably on disk now
    - Ok because of 10y experience with detector to know what we need
  - For HL-LHC, new detectors may require more time with low-level information
- Best cache: all the columns you need, none you don't
  - Different set of columns needed for different PDs, analyses
  - Not all rows read if filtering (skimming)
  - **How much better than PD x tier granularity we have now?**





https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults

Access patterns

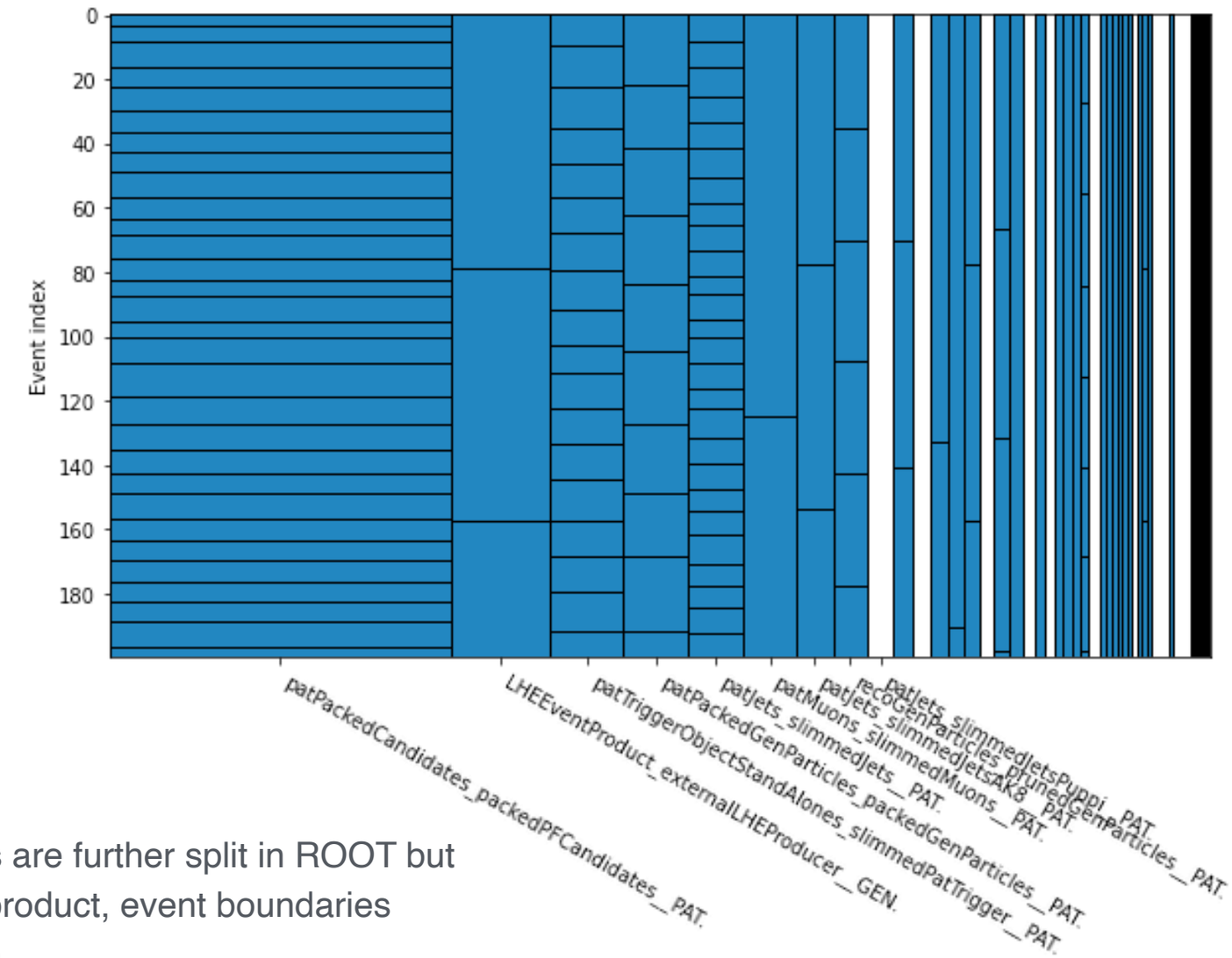
- FrameworkJobReport.xml can help answer this question
  - Files opened, number of events read per data-product, storage timings
  - Sadly we don't collect these for all jobs, just some summary stats
  - Asked around for a few examples (more would be nice), below is one
- Basket = ROOT TBasket
  - Smallest unit of data access (if file seek allowed)

Branch access stats for PFNano (MiniAOD input)

branch	events read	compressed bytes / evt	est. baskets read
patPackedCandidates_packedPFCandidates__PAT.	3600	19864	518
LHEEventProduct_externalLHEProducer__GEN.	3597	5782	46
patTriggerObjectStandAlones_slimmedPatTrigger__PAT.	3597	4343	323
patPackedGenParticles_packedGenParticles__PAT.	3600	3739	172
patJets_slimmedJets__PAT.	3600	3254	472
patMuons_slimmedMuons__PAT.	3600	3091	28
patJets_slimmedJetsAK8__PAT.	3600	2141	46
recoGenParticles_prunedGenParticles__PAT.	3600	2033	99
patJets_slimmedJetsPuppi__PAT.	0	1447	0
EcalRecHitsSorted_reducedEgamma_reducedEBRecHits_PAT.	3597	1128	51
patElectrons_slimmedLowPtElectrons__PAT.	0	1125	0
patJets_slimmedJetsAK8PFPuppiSoftDropPacked_SubJets_PAT.	3600	941	27
patElectrons_slimmedElectrons__PAT.	3600	934	19
patTaus_slimmedTaus__PAT.	3600	919	46
patTaus_slimmedTausBoosted__PAT.	0	882	0
Other (size-weighted)	2180	12670	484

Access patterns

- Another view of the same information: each box is a basket*
  - Basket size: larger = better compression but less chance of skipping basket if skim
  - Location of basket is in TTree header: lots of metadata
  - Redraw boxes so they factorize across row*column? Group skinny boxes together?



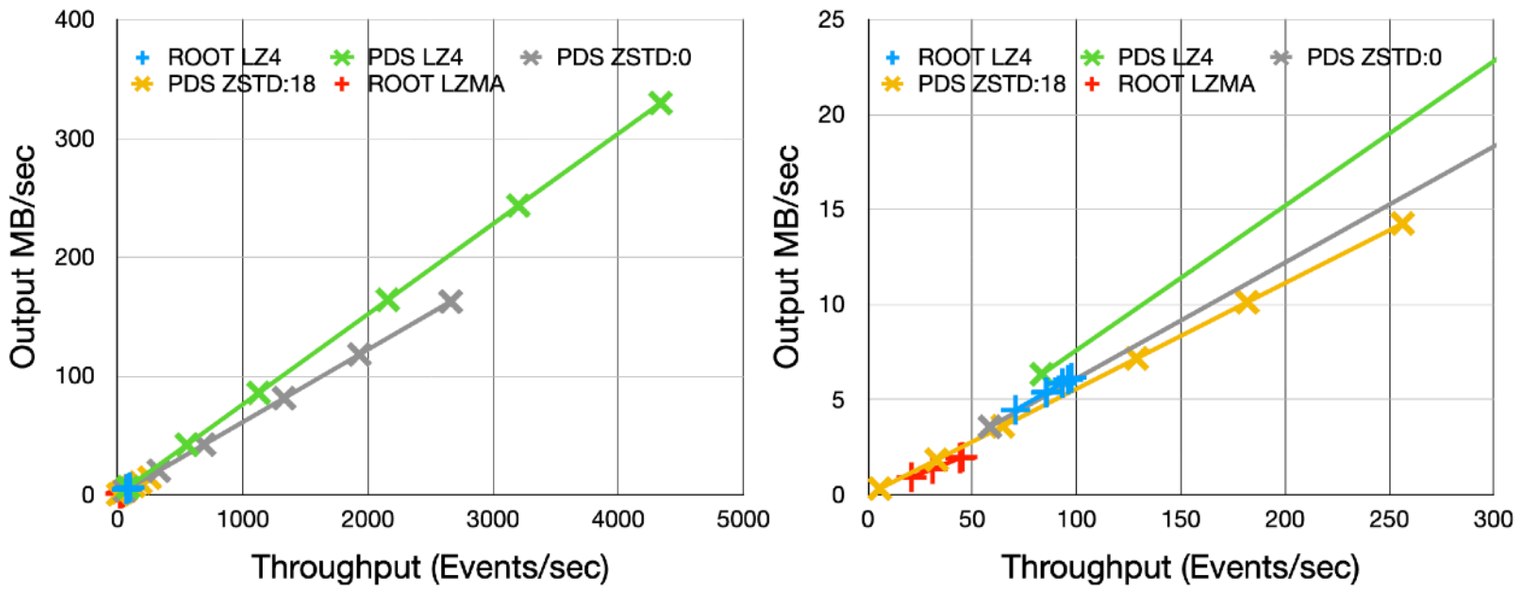
* Schematic: most branches are further split in ROOT but accessed as one CMSSW product, event boundaries approximated (avg. + noise)

Fermilab

I/O performance

- CCE-IOS project to evaluate CMSSW (& other) I/O performance
  - Output module is a bottleneck for CMSSW above 8 threads
  - With (in development) ROOT concurrent I/O, issue may go away
  - However, still worth investigating performance of writing to many objects vs. one file
- Framework to evaluate alternative I/O strategies
  - https://github.com/hep-cce2/root_serialization
  - Easy to add new output modules, simulate event processing, and test I/O

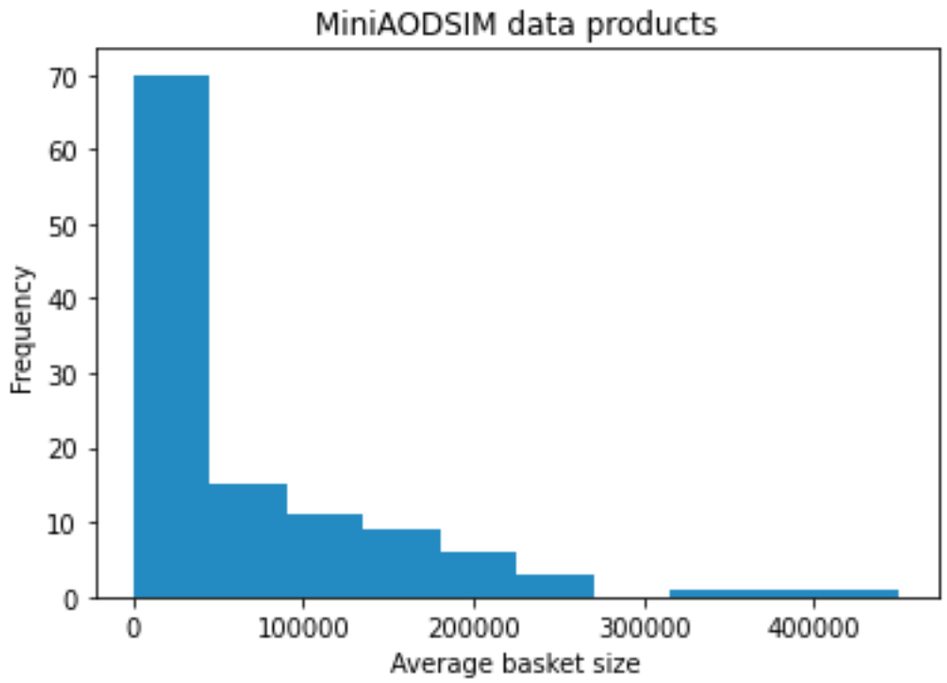
MINIAOD Output Rate vs Throughput



Fermilab

Why object stores?

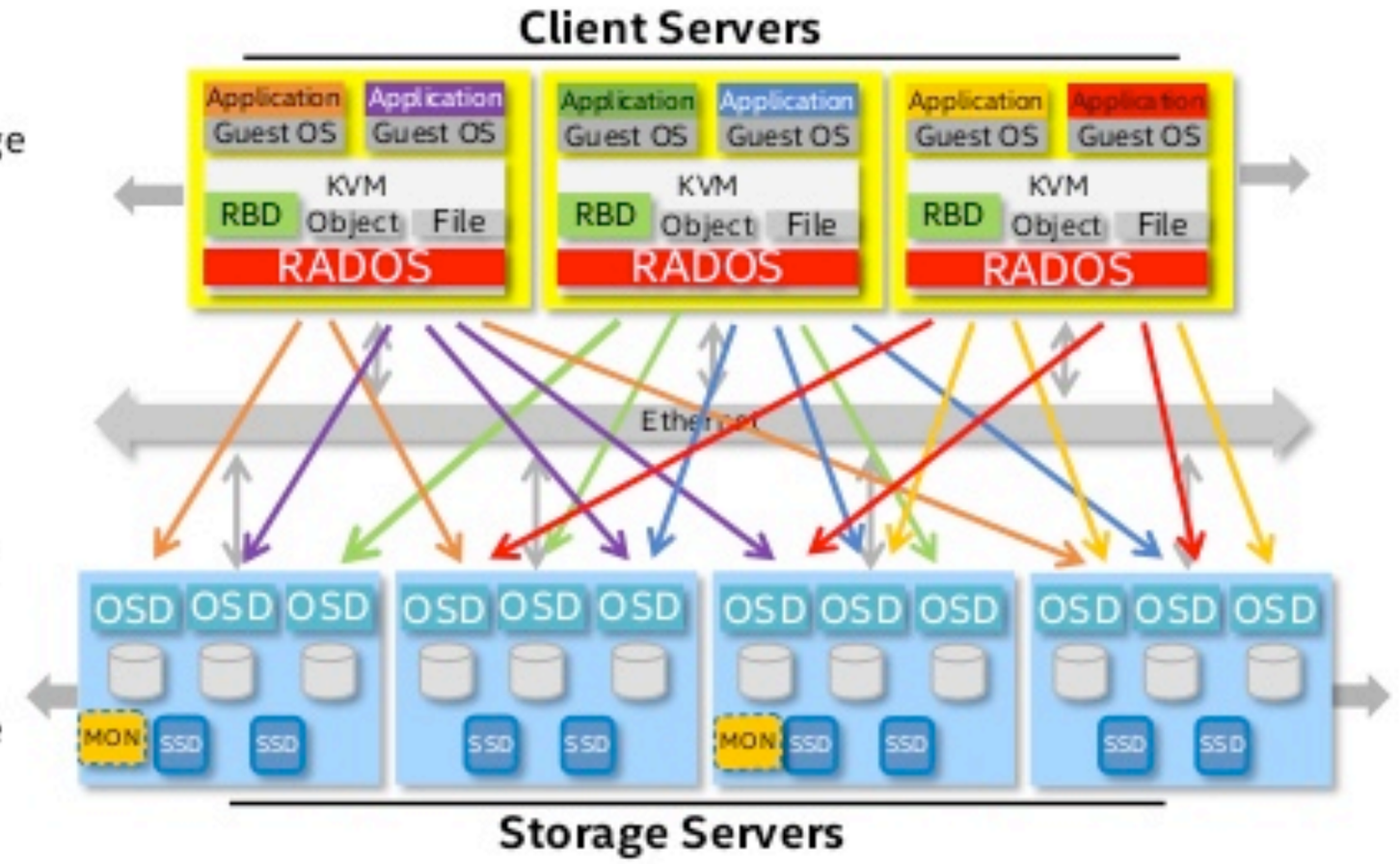
- Suppose we want a 1PB fine-grained cache of MiniAOD
  - 50k/event -> 20B events
  - PFNano example read 2330 baskets for 3600 events -> 13B basket blobs
    - Probably a lot fewer if we group tiny columns together
  - So we need something that can read and write 10 billion binary blobs of ~100kB
- Filesystems can store 10B files, but
  - Metadata overhead significant for many small files
  - We get a lot of features we do not need: seeks, per-file permissions, partial writes
  - Scanning, moving, synchronizing, recovery… a pain
- Ceph object store: 10B is no problem
  - Magic bullet: calculated placement
    - Like a hash, can be run client-side
    - Inputs: object key and cluster topology
    - Output: what server, disk holds data
  - Other benefits:
    - Erasure coding (also with CephFS)
    - S3 access protocol
    - Cache tiering



Ceph architecture

Ceph Cluster Overview

- **Ceph Clients**
  - Block/Object/File system storage
  - User space or kernel driver
- **Peer to Peer via Ethernet**
  - Direct access to storage
  - No centralized metadata = no bottlenecks
- **Ceph Storage Nodes**
  - Data distributed and replicated across nodes
  - No single point of failure
  - Scale capacity and performance with additional nodes



Ceph scales to 1000s of nodes

IDF15

9

https://insujang.github.io/2020-08-30/introduction-to-ceph/

Plan

- Benchmark ceph, understand performance scaling and limits
- Implement object I/O in CCE-IOS benchmark tool, eventually in CMSSW
- Benchmark I/O performance and space efficiency with object storage
- Move some interesting data into system, run real workflows
  - Best use case: analyses that used to read MiniAOD+AOD two-file input
- Implement ServiceX module to produce custom NanoAOD?
  - Holy grail: add a column to a dataset without re-computing the rest