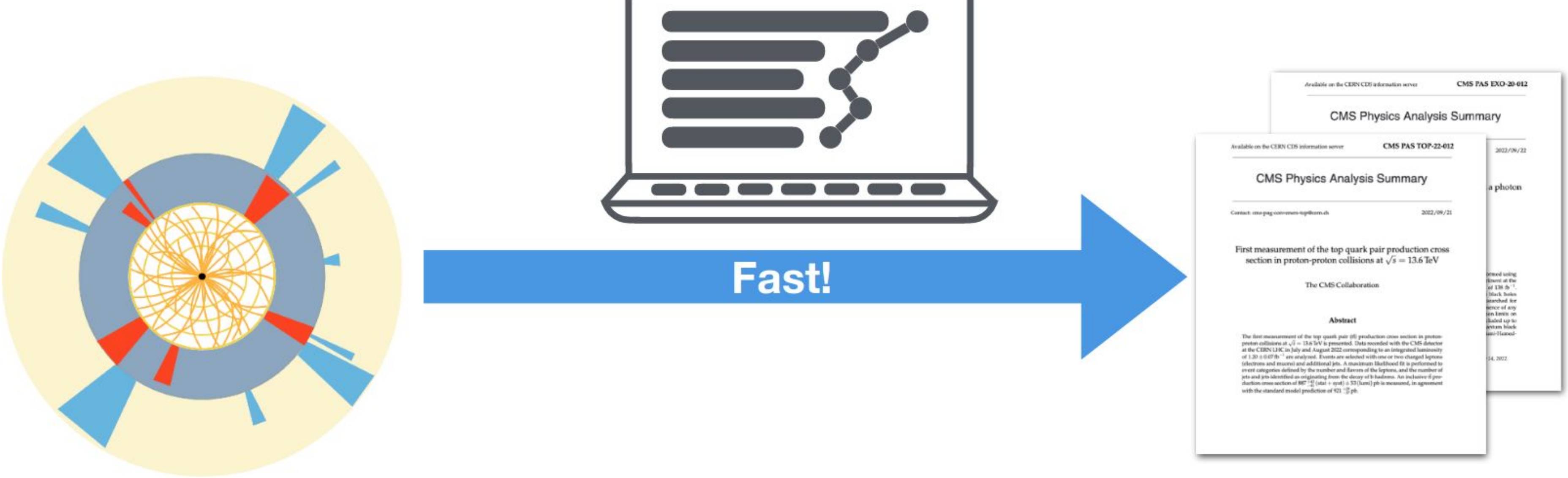
IDIUM



Transferable Improved Analyses Turnaround through Intelligent Caching and Optimized Resource Allocation

Svenja Diekmann, Niclas Eich, Martin Erdmann, Peter Fackeldey, Benjamin Fischer, Dennis Noll, Yannik Rath

CHEP 2023 - Track 1
May 9th 2023

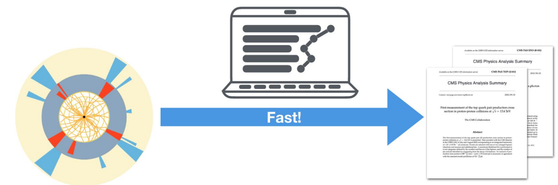
SPONSORED BY THE
Federal Ministry of Education and Research

RWTH AACHEN UNIVERSITY

Outline

Motivation

- Small time-to-insight drives high physics potential
- Improved throughput important for HL-LHC



Computing Stages & Enhancements

1. Ingestion: i.e. reading from file
   ➜ Caching (intelligent, transparent)
2. Processing: computations & filtering
   ➜ Compute Offloading
3. Aggregation: counting & histrogramming
   ➜ Memory Offloading

} enable more precise Resource Allocation

Context

- small institute cluster
  ➜ need to make effective use
- our implementation: coffea + Dask + HTCondor
  ➜ but, principles apply generally

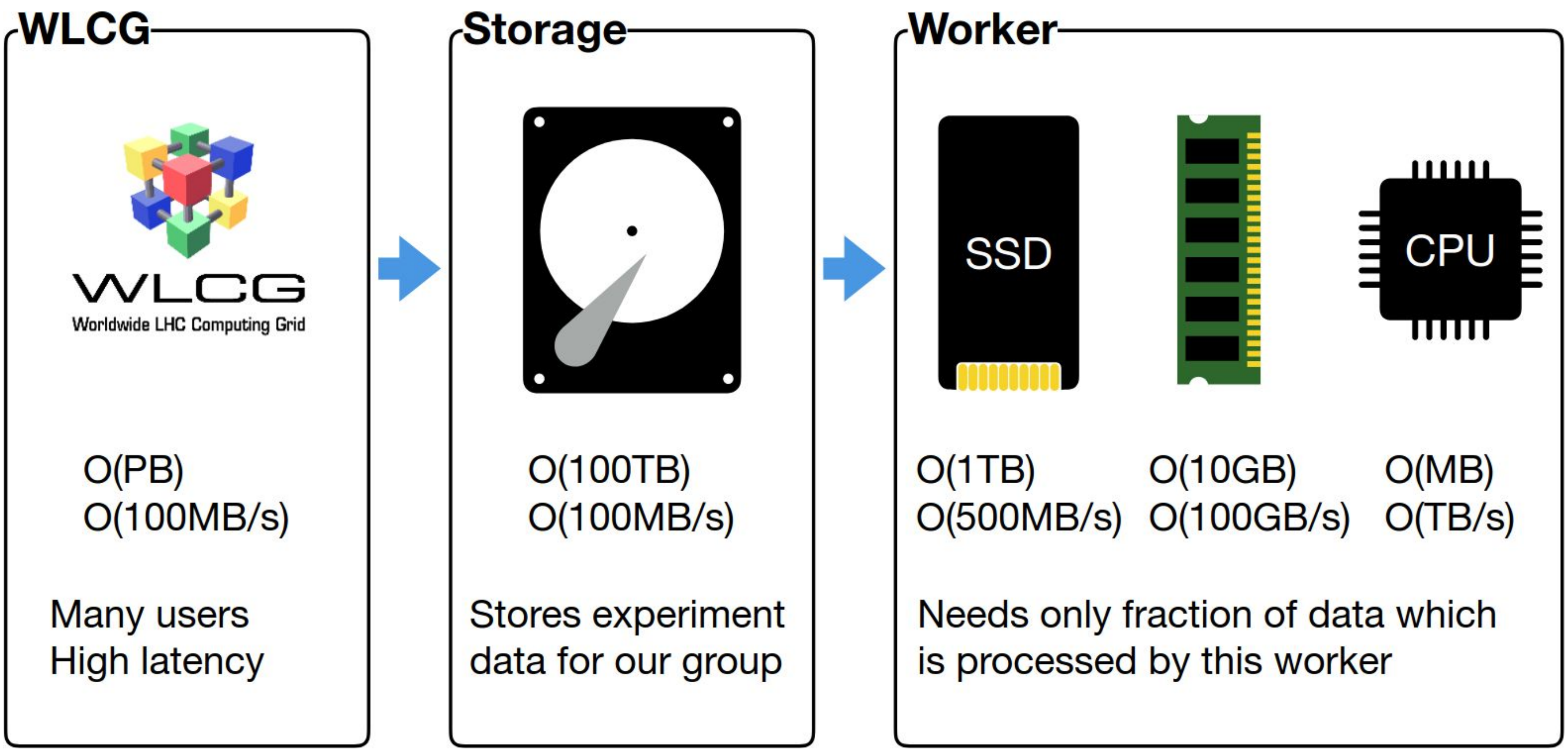
coffea

dask

HTCondor
High Throughput Computing

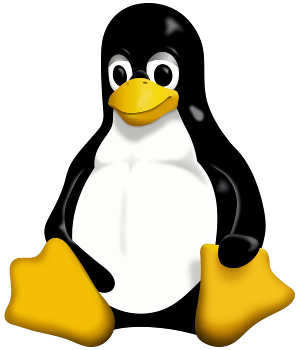
Caching: Storage Hierarchy

- Critical bottleneck: streaming data to processing elements
- Available: various storage types & locations
- Solution: two tiered persistent caching
  1. central network storage (NFS)
  2. on Worker SSD (FSCache)

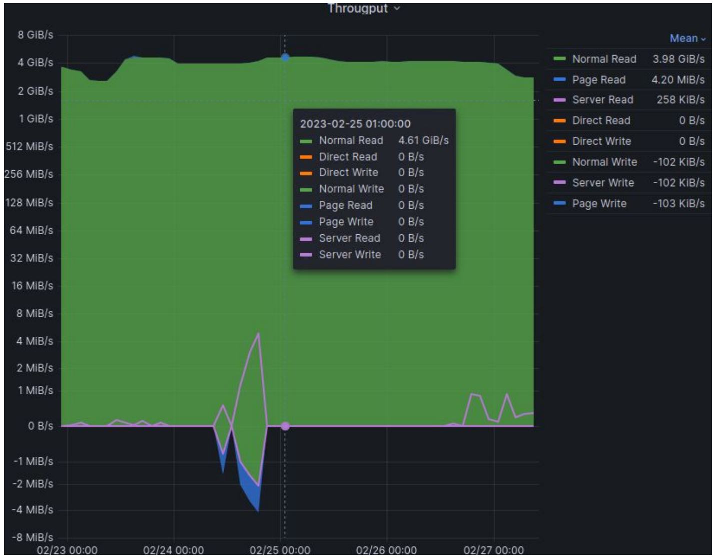


Worker Caching (FSCache)

RWTH AACHEN UNIVERSITY
Benjamin Fischer, May 9th 2023



- Use worker's SSDs for caching
- Software implementation: FSCache + cachefilesd
  - Transparently caches:
    - file access (read & write)
    - enabled per NFS-mount
  - Granularity: Page size (4kB)
  - Strategy: Least recently used (LRU)
    - prone to trashing
  - Part of Linux Kernel (ideally ≥5.4)
- Great for caching of:
  - software, e.g. conda environments
  - event files, e.g. NanoAOD's .root files
  - DNN training data, e.g. .npy files

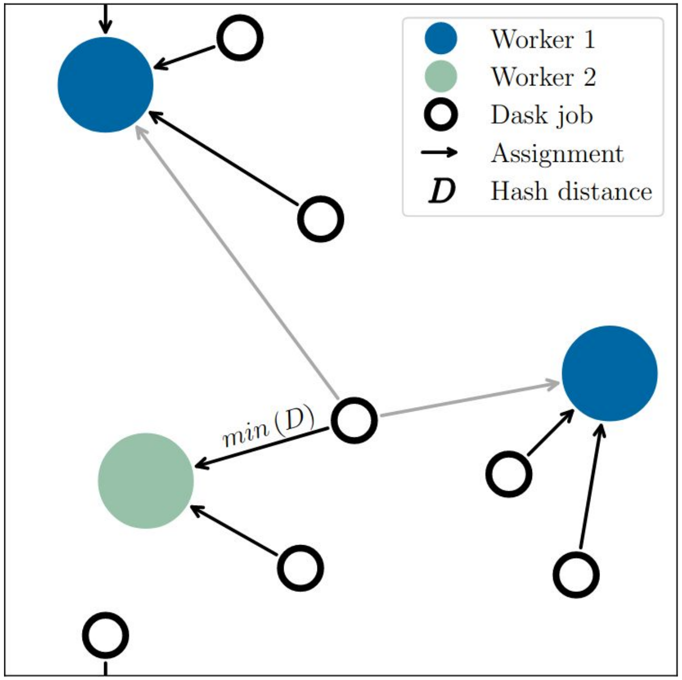


Performance example: DNN training

- serves sustained 4.6 GB/s across whole cluster
- comparison: same bandwidth via central server far more difficult

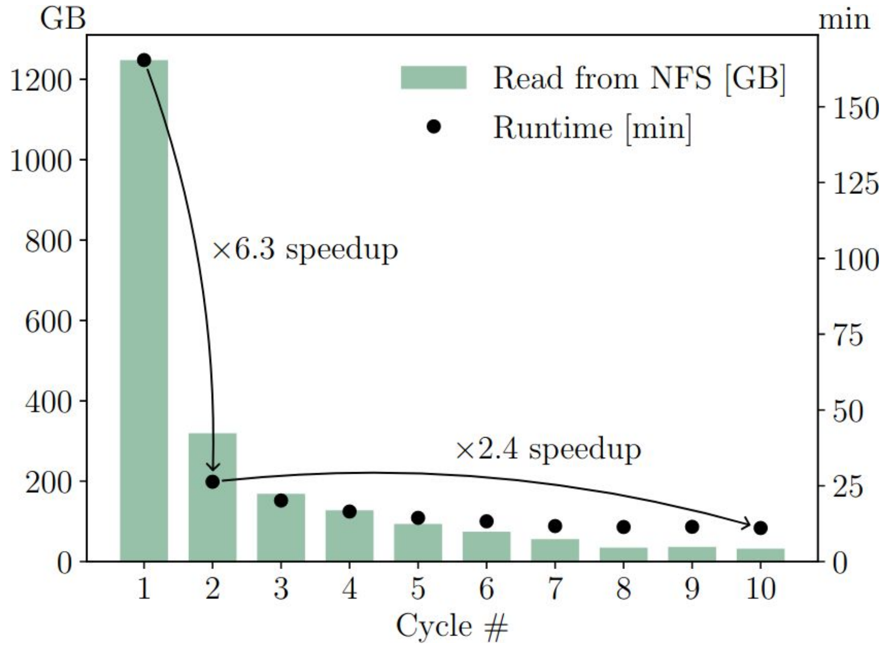
RWTHAACHEN UNIVERSITY
Benjamin Fischer, May 9th 2023

- Aim: ensure persistent Worker ↔ Job assignment
  - translates to FSCache-Instance ↔ Input-Data
    - ➜ minimizes cache thrashing due to LRU strategy
- Assign reproducible identifiers to: Workers & Jobs
  - using cryptographic hash function ➜ uniformly distributed
  - produces 64-dimensional embedding: $\mathbb{x}^{64}$ with $\mathbb{x} \in [0\ldots255]$
  - interpreted as position vectors
- Assign Job to closest Worker (L^2 distance)
- Extra tricks for Workers:
  - multiple distinct positions per Worker
    - ➜ more uniform distribution despite the low number of total Workers
  - additional distance factor per Worker
    - ➜ incorporates varying computational throughput
- Resilience: handles perturbations well
  - needed for changing Workers/Jobs



Cache Benchmark 2207.08598

- Pure read-only task run 10 times (cycles) with 220 workers (no computations)
  - isolate & highlight IO performance
- "Processed" Data: Higgs pair production analysis
  - 1.3 TB, 10^9 events, 120 columns, nanoAOD
  - read-optimised compression algorithm (Z-std, 10)
- Results: cache effect well visible
  - gradual convergence to perfection due "work stealing"
  - runtime lower bound: CPU → IO Bottleneck overcome



Compute Offloading

RWTH AACHEN UNIVERSITY
Benjamin Fischer, May 9th 2023

Idea

- offload onto specialized hardware i.e. GPU
- works well with heavy workloads:
  - esp. DNN evaluation
  - possibly even fits/ME-calc.

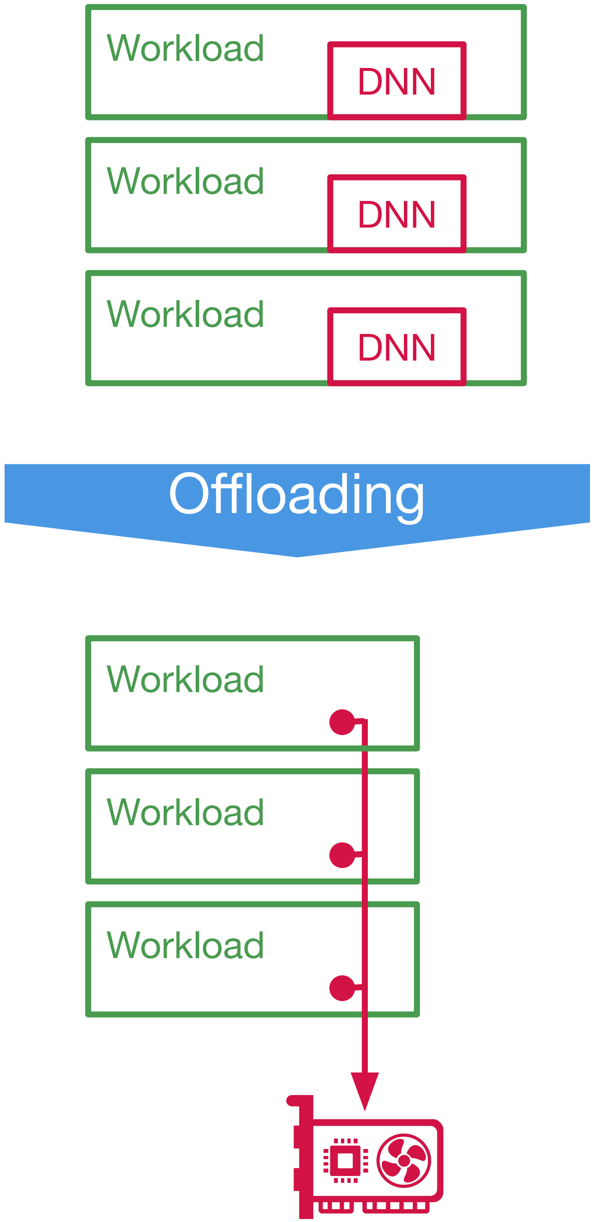
Advantages

- batching possible/needed
  - favors columnar data processing
  - automatic batching possible
- enables parallel/async processing
- also has Memory Offloading benefits

Implementation

- using Tensorflow Modelserver
- single low-end GPU sufficient
  - 10^9 events/hour

TensorFlow



RWTH AACHEN UNIVERSITY
Benjamin Fischer, May 9th 2023

Antipattern

excessive buffering of intermediate output
➜ footprint of workload varies a lot
➜ poor resource allocation use

Histograms: pathological affected

- esp. when multiple workloads produce same histograms to be summed later

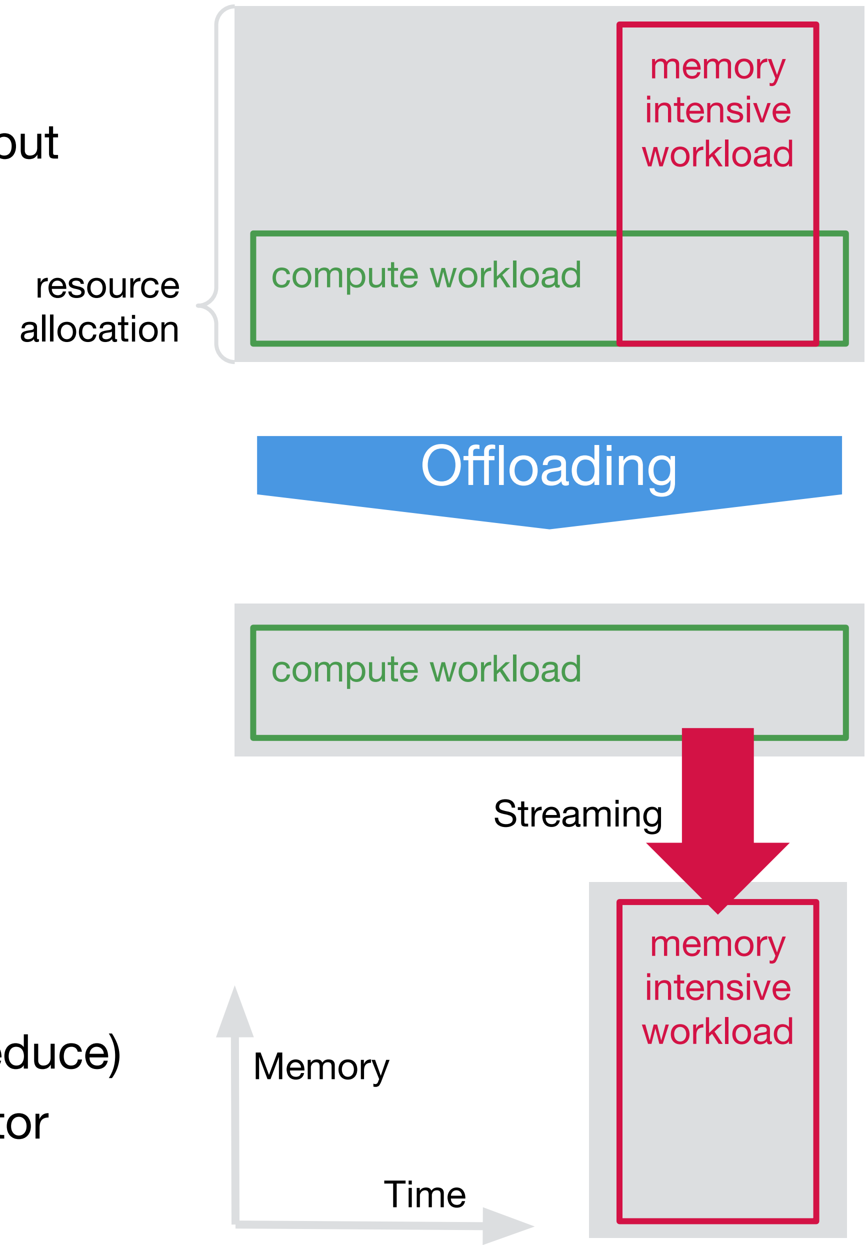
➜ unnecessary copies

Solution: eagerly aggregate outputs

- opportunity for centralization

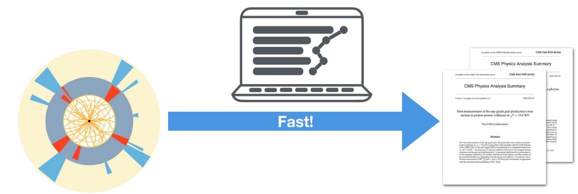
Implementation

- data streamed (decoupled from MapReduce)
- transparent via custom coffea Aggregator
- communication via Dask (Actors)



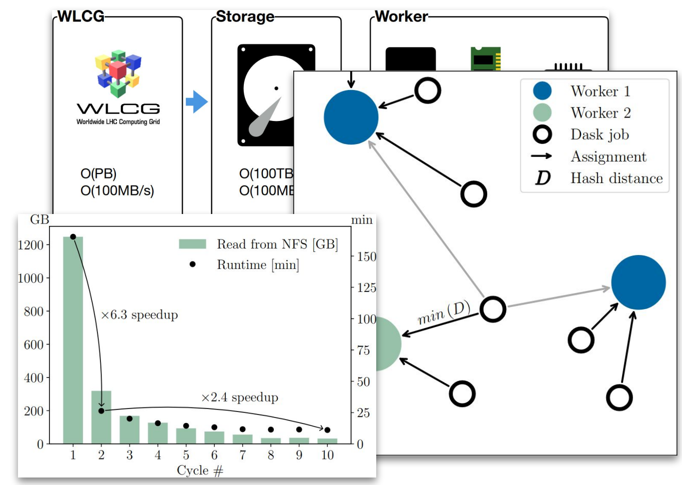
Benjamin Fischer, May 9th 2023

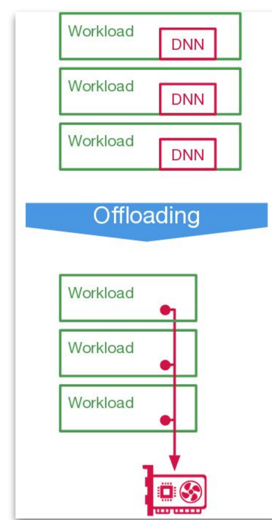
- Context: small institute cluster
- Goal: increase utilization/throughput

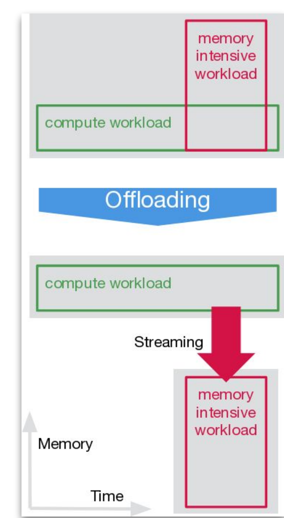


Enhancements

- Caching ➜ decrease IO stall 2207.08598
  - opportunistic (use what's available)
  - tiering (storage distribution)
  - ease of use (fully transparent)
  - resilience (thrashing mitigation)
- Offloading ➜ improve utilisation
  - ➢ Compute ~: DNNs onto central GPU
  - ➢ Memory ~: histograms into dedicated job
  - better fitting allocation possible







Transferable

- principles apply in general
- also relevant for e.g. XCache, RDataFrame, …

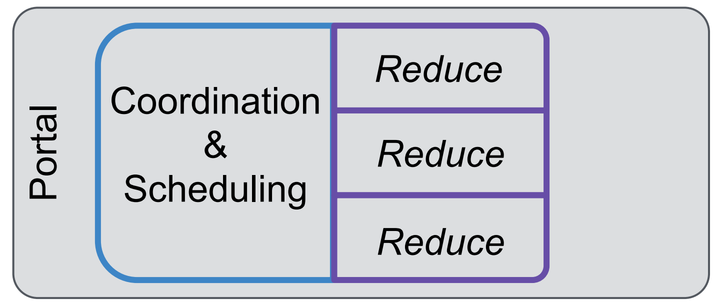
Backup

MapReduce + Dask + HTCondor

- MapReduce via Dask via HTCondor
  - enables dynamic workload distribution across resources as they get available

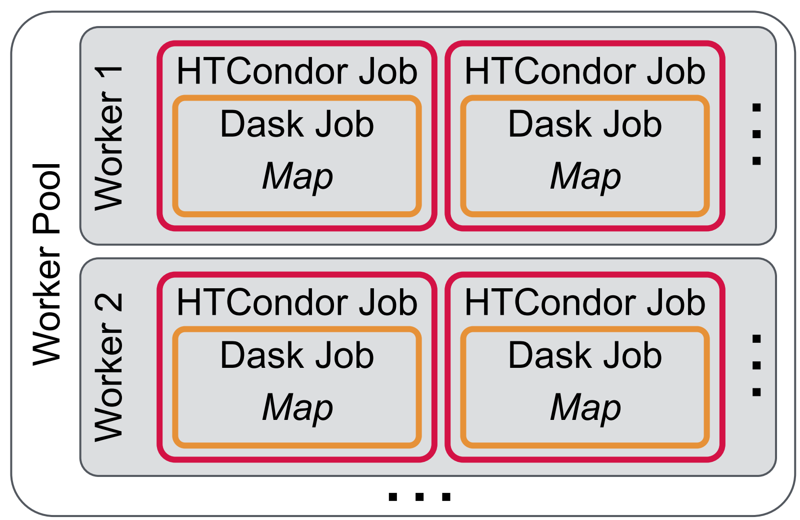
Portal node (direct user access):

- central coordination & scheduling
- runs *Reduce* tasks:
  - merges outputs (e.g. histograms)
  - in- & output size highly variable



Worker pool (via HTCondor):

- variable availability
- runs *Map* tasks:
  - the actual processing workload (e.g. tuples ➜ histograms)
  - fairly homogeneous footprint ➜ ideal for statically booked Job resources



Central Network Storage

RWTH AACHEN UNIVERSITY
Benjamin Fischer, May 9th 2023

- Keep local experiment data copies:
  - Easy and reliable access (no timeouts, credentials, ...)
  - Direct connection to worker nodes (low latency, 10GBit)
- Storage qualities:
  - /home: User homes (mirrored, backup, low latency)
  - /store: Experiment data (mirrored, high capacity)
  - ➜ /scratch: Experiment data (for copies i.e. reproducible/redownloadable)
    - optimized for high read throughput
    - RAID0: not mirrored, but striped (across multiple HDDs)