Opportunistic data locality for end user data analysis

Max Fischer, Eileen Kühn, Christoph Heidecker, Matthias Schnepf, Manuel Giffels
CHEP 2016
Context: HEP End User Data Analysis

- Hierarchical, iterative workflows
  - Reduction of data size
  - Increase of iterations
  - Dedicated processing environments

![Graph showing the relationship between data size and revisions.

For Dataset, Skim, Analysis, and Plot, the data size decreases as the number of revisions increases.]
Context: HEP End User Data Analysis

- Hierarchical, iterative workflows
  - Reduction of data size
  - Increase of iterations
  - Dedicated processing environments

- Data intense analyses on Tier 3
  - Standard batch systems and fileservers
  - Extraction of observables from optimized data sets/formats
Context: HEP End User Data Analysis

- Hierarchical, iterative workflows
  - Reduction of data size
  - Increase of iterations
  - Dedicated processing environments

- Data intense analyses on Tier 3
  - Standard batch systems and file servers
  - Extraction of observables from optimized data sets/formats

- Usage suitable for caching
  - Repeated processing of same input
  - Strongly dependent on input rate
Coordinated Caching: Overview

- Caching between batch system and data sources
  - Consumer focused caching
  - Provides partial data locality
Coordinated Caching: Overview

- Caching between batch system and data sources
  - Consumer focused caching
  - Provides partial data locality

- Abstracts cache to batch system scale
  - Utilize meta-data of entire user workflows
  - Works on files used by jobs
Coordinated Caching: Overview

- Caching between batch system and data sources
  - Consumer focused caching
  - Provides partial data locality
- Abstracts cache to batch system scale
  - Utilize meta-data of entire user workflows
  - Works on files used by jobs
- Implementation at host granularity
  - Array of individual caches on worker nodes
  - Caches coordinated by global service
  - Some glue for data locality…
Coordinated Caching: Data Availability

- Distributed caching complicates cache access
  - Data cached anywhere (cache hit rate)
  - Data local on job host (local hit rate)
Coordinated Caching: Data Availability

- Distributed caching complicates cache access
  - Data cached anywhere (cache hit rate)
  - Data local on job host (local hit rate)

- Schedule Jobs to input data location
  - Unscheduled hit rate limited to \(~1/N_{\text{worker}}\)
  - Data location published to batch system
Coordinated Caching: Data Availability

- Distributed caching complicates cache access
  - Data cached anywhere (cache hit rate)
  - Data local on job host (local hit rate)

- Schedule Jobs to input data location
  - Unscheduled hit rate limited to $\sim 1/N_{\text{worker}}$
  - Data location published to batch system

- Place data to match workflows
  - Jobs require groups of files
  - Placement uses observed data splitting
Coordinated Caching: Throughput Simulation

- Batch system throughput simulation
- Setup of KIT Tier3
- Parameters: local hit rate, \( N_{\text{worker}} \)

![Diagram of Coordinated Caching](image)
Coordinated Caching: Throughput Simulation

- Batch system throughput simulation
  - Setup of KIT Tier3
  - Parameters: local hit rate, $N_{\text{worker}}$

- Caching allows horizontal scaling
  - Throughput scales with workers…
  - …if jobs are scheduled to data
Coordinated Caching: Throughput Simulation

- Batch system throughput simulation
  - Setup of KIT Tier3
  - Parameters: local hit rate, $N_{\text{worker}}$

- Caching allows horizontal scaling
  - Throughput scales with workers…
  - …if jobs are scheduled to data

- Perfect hit rate not ideal
  - Leverage remote I/O
  - Potential to…
    - Use simple algorithms
    - Increase effective cache size
HTDA Batch System Extension

High Throughput Data Analysis

Diagram showing the flow from Submitter, through Jobs, to Scheduler, and then to Workers.
HTDA Batch System Extension

High Throughput Data Analysis

- Caches maintain data copies on worker nodes
Caches maintain data copies on worker nodes
Locator provides locality information for jobs
Caches maintain data copies on worker nodes
Locator provides locality information for jobs
Coordinator schedules files for caching on nodes
Prototype Batch System

- Extends HTCondor setup
- Static, opportunistic and HTDA nodes in same cluster
- 5 HTDA worker nodes
- á 500 GB SSD cache
- 6 fileservers
Experience: Batch System Integration

- Hooks on submission hosts via `job_router`
  - Integrates directly into batch system
  - Efficient push instead of pull behavior
  - Constraint of 1 active route (service) per job
Experience: Batch System Integration

- Hooks on submission hosts via `job_router`
  - Integrates directly into batch system
  - Efficient push instead of pull behavior
  - Constraint of 1 active route (service) per job

- Only job meta-data exchanged
  - Job features from HTCondor
  - Placement information from HTDA
Experience: Batch System Integration

- Hooks on submission hosts via \textit{job\_router}
  - Integrates directly into batch system
  - Efficient push instead of pull behavior
  - Constraint of 1 active route (service) per job

- Only job meta-data exchanged
  - Job features from HTCondor
  - Placement information from HTDA

- Efficient interface to HTCondor
  - Selection/tracking handled by HTCondor
  - Hook skips any meaningless updates
  - Arbitrary number of untracked jobs
Experience: User Workflows

- Benchmark workflow: CMS calibration
- ROOT n-tuple analysis
- 400 GB LHC run1 input data
- Notable improvement
Experience: User Workflows

- Benchmark workflow: CMS calibration
  - ROOT n-tuple analysis
  - 400 GB LHC run1 input data
  - Notable improvement

- Used for LHC run2 user analyses
  - Single patch to submission tool
  - Fully transparent in regular cluster
  - Non-intrusive to regular operation
Experience: HTDA Middleware Performance

- Mature prototype implementation
- Stable operation for 6+ months
- Worker CPU/RSS overhead negligible

|                | CPU  | RSS  |
|----------------|------|------|
| Cache          | 3.5% | 120 MB |
| Locator        | 1.0% | 60 MB  |
| Coordinator    | 14.1%| 1 GB  |
Experience: HTDA Middleware Performance

- Mature prototype implementation
  - Stable operation for 6+ months
  - Worker CPU/RSS overhead negligible

- Mixed experiences with SL6 (2.6 kernel)
  - Similar analysis (ROOT) performance as on 3.X kernel systems
  - Availability reduced by unstable AUFS 2.X (for cache access)

|                  | CPU (%) | RSS     |
|------------------|---------|---------|
| Cache            | 3.5     | 120 MB  |
| Locator          | 1.0     | 60 MB   |
| Coordinator      | 14.1    | 1 GB    |
Experience: HTDA Middleware Performance

- Mature prototype implementation
  - Stable operation for 6+ months
  - Worker CPU/RSS overhead negligible

- Mixed experiences with SL6 (2.6 kernel)
  - Similar analysis (ROOT) performance as on 3.X kernel systems
  - Availability reduced by unstable AUFS 2.X (for cache access)

- Open issues: no showstoppers
  - Deliberate cleanup of meta-data and file reallocation
  - Tweaks and optimizations

|                | CPU  | RSS  |
|----------------|------|------|
| Cache          | 3.5% | 120 MB |
| Locator        | 1.0% | 60 MB  |
| Coordinator    | 14.1%| 1 GB  |
Summary

- Coordinated Caches for Batch Systems
  - Array of caches on worker nodes
  - Coordination by global service
  - Targets input files of user workflows

- Prototype Implementation: HTDA
  - Proof of principle, all major features covered
  - Room for improvements and extensions
  - Already considerable performance improvements

- Applicable to other setups
  - Shared caches via parallel filesystems
  - Cache-only Tier3 without dedicated storage
  - Persistent cache for opportunistic resources
BACKUP
Cache Content Access

- Cache node stages/unstages files according to coordinator request

Diagram:

- SSD
- NFS
- HTDA

Diagram shows the interaction between SSD, NFS, and HTDA for cache content access.
Cache Content Access

- Cache node stages/unstages files according to coordinator request
- Union File System provides transparent cache access for users
Cache Content Access

- Cache node stages/unstages files according to coordinator request
- Union File System provides transparent cache access for users

- Lightweight cache access ensures optimal performance

**Graph: ROOT N-Tuple Analysis**

- Disk/Net Read (MB/s)
- Concurrent Processes

**Systems:**
- UFS
- SSD
- NFS

**Legend:**
- SSD
- AUFS
I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test

**ROOT N-Tuple Analysis**

- Disk/Net Read (MB/s) vs. Concurrent Processes (+48)
- CPU Utilisation (%) per Job vs. Concurrent Processes (+48)

2006 Tier2 CPU capacity
I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test

HDDs limited on concurrent accesses
I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test

ROOT N-Tuple Analysis

HDDs limited on concurrent accesses
SSDs exploit full system capacities