You may also be interested in:

Evaluation of Apache Hadoop for parallel data analysis with ROOT
S Lehrack, G Duckeck and J Ebke

On the Factor Refinement Principle and its Implementation on Multicore Architectures
Md Mohsin Ali, Marc Moreno Maza and Yuzhen Xie

Implementation of a solution Cloud Computing with MapReduce model
Chalabi Baya

Running a typical ROOT HEP analysis on Hadoop MapReduce
S A Russo, M Pinamonti and M Cobal

ALICE HLT TPC Tracking of Pb-Pb Events on GPUs
David Rohr, Sergey Gorbunov, Artur Szostak et al.

Experience with Intel's Many Integrated Core architecture in ATLAS software
S Fleischmann, S Kama, W Lavrijsen et al.

Forming an ad-hoc nearby storage, based on IKAROS and social networking services
Christos Filippidis, Yiannis Cotronis and Christos Markou

Exploiting the ALICE HLT for PROOF by scheduling of Virtual Machines
Marco Meoni, Stefan Boettger, Pierre Zelnicek et al.

Performance optimisations for distributed analysis in ALICE
L Betev, A Gheata, M Gheata et al.

Data Locality via Coordinated Caching for Distributed Processing

M Fischer, E Kuehn, M Giffels, C Jung

Karlsruhe Institute of Technology, Steinbuch Centre for Computing,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

E-mail: {`max.fischer, eileen.kuehn, manuel.giffels, christopher.jung`}@kit.edu

Abstract. To enable data locality, we have developed an approach of adding coordinated caches to existing compute clusters. Since the data stored locally is volatile and selected dynamically, only a fraction of local storage space is required. Our approach allows to freely select the degree at which data locality is provided. It may be used to work in conjunction with large network bandwidths, providing only highly used data to reduce peak loads. Alternatively, local storage may be scaled up to perform data analysis even with low network bandwidth.

To prove the applicability of our approach, we have developed a prototype implementing all required functionality. It integrates seamlessly into batch systems, requiring practically no adjustments by users. We have now been actively using this prototype on a test cluster for HEP analyses. Specifically, it has been integral to our jet energy calibration analyses for CMS during run 2. The system has proven to be easily usable, while providing substantial performance improvements.

Since confirming the applicability for our use case, we have investigated the design in a more general way. Simulations show that many infrastructure setups can benefit from our approach. For example, it may enable us to dynamically provide data locality in opportunistic cloud resources. The experience we have gained from our prototype enables us to realistically assess the feasibility for general production use.

1. Introduction

End user data analysis tasks in HEP are commonly processed by hundreds of jobs on a batch cluster, reading data over network from file servers. As we have shown in earlier work [1, 2], an analysis on a modern institute cluster easily saturates network capacity. With moving simulation jobs to opportunistic resources [3], we expect saturation from analysis jobs to be more frequent. To enable efficient analyses in the future, we therefore investigated data locality as a means to eliminate dependency on network resources.

Data locality approaches reduce overall remote I/O by executing jobs as close as to their input data as possible. Ideally, the machine executing a job and hosting its data are the same. Several frameworks such as Hadoop [4] already provide data locality based processing, and have proven the feasibility of this approach.

However, we have found such frameworks to be inadequate for end user analyses. For example, the extent of software modifications required would effectively eliminate portability to and from other infrastructure. Thus, we have developed an alternate approach to data locality that integrates into regular batch processing.

Our approach uses coordinated caches to provide data locality for a fraction of data. This exploits that at any time, only a few sets of data contribute to overall throughput (see Figure 1). By eliminating remote accesses to them, network capacity remains available for less frequently used data. This is a similar strategy to the current mixture of simulation and analysis tasks.

We have implemented a prototypical middleware [5], targeting the HTCondor batch system [6]. This prototype is deployed on a portion of the local KIT CMS analysis groups' batch system. It has since been successfully used for the CMS Jet Energy Scale calibration analyses performed at KIT.

In this paper, we focus on discussing advantages and disadvantages of our approach and prototype. Section 2 details the features inherent to our approach in general. In section 3, we discuss our current prototype implementation. Finally, Section 4 provides a short conclusion.
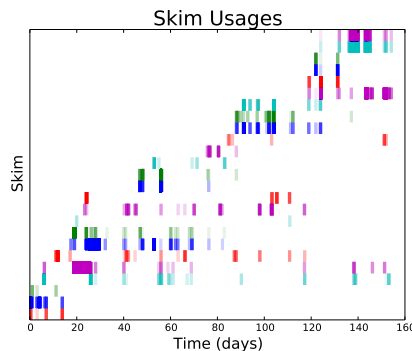

Figure 1: Read accesses from jobs to skim versions: Over time, users create new skims for their analyses. Only some of these are used frequently, however. Often, it takes several intermediate iterations before a skim is replaced.

2. Coordinated Caching in Batch Systems

Individual features of our approach have been subject of past research. Coordinated caching has been shown to be effective in distributed systems, e.g. web services [7]. Applicability of data locality frameworks for HEP has been investigated in abundance [8, 9]. Limited data locality via a middleware has been attempted using cache servers [10]. Other caches for batch processing provide applications used across several jobs [11, 12] Our work is set apart mainly by the scope in terms of size and subjects of caching.

2.1. Scope and Granularity

Our approach is to have a single cache target the batch system as a data consumer. This sets us apart from coordinated caches that target data providers such as web services. The systen itself can be compared to a scaled up operating system page cache. A page cache targets *applications* accessing *blocks* via *read* system calls to process *files*. Our cache targets *workflows* accessing *files* via *jobs* to process *datasets*. Under the hood, the system is composed of several caches, one on each worker node. These are joined together by a coordination service.

The biggest advantage is the scaled up decision layer, selecting files for caching. Even in a small cluster, using a few cores for managing the caches is negligible overhead. Likewise, storing file meta-data in the scale of MB is negligible compared to file sizes at the scale of GB. Since jobs operate on the scale of minutes to hours, the system does not need to respond any faster. This allows for sophisticated caching logic.

The biggest challenge originates from our cache volume being actually several distinct volumes. Coordinating these is not a technical challenge, but a scheduling problem. Since we want to avoid remote accesses, jobs and data must be closely aligned.

2.2. Data Selection and Hit Rates

Using coordinated caches for data locality adds another dimension to data handling. It is not sufficient to have a file anywhere in the cache. Instead, it must be available on the host jobs are trying to access it from. This is the key reason why coordination is required for distributed caching of unique input to be effective. If files and jobs were placed randomly, the chance of a file

and corresponding job being on the same worker node is inversely proportional to the number of nodes. When a job requires several files, the expected fraction of files locally available converges to the inverse number of nodes. Even for small clusters, this makes the impact of uncoordinated caching negligible.

In most setups, network throughput still is substantial. To maximize overall throughput, it is best not to read everything from cache, but instead read some data over network. This can easily be achieved by caching only a portion of the data deemed relevant. There are two extremes to this: On one end, caches provide just enough data to not overburden the network. This maximizes cache volume, as only a fraction of each dataset must be provided. On the other end, caches provide as much data as possible, freeing the maximum of network resources. This is optimal if there are many workflows unsuitable for caching, as these can use the network fully.

2.3. User Workflow Integration

Being designed as a cache, our system relies on intercepting access requests. This contrasts with dedicated data locality solutions, which require explicit requests to the middleware.

As we treat the entire job as an access, requests are intercepted at different points. On the one hand, jobs are intercepted as meta-data in the batch system. This provides extensive information, e.g. estimated runtime. On the other hand, the file accesses of jobs executing on worker nodes are intercepted. This implements the actual rerouting to local data.

Intercepting requests has the advantage of being transparent to end users. It does not make a technical difference to jobs whether our system is present. The only notable difference is an increase in performance if files are provided from our cache. This ensures optimal portability.

The downside of a transparent system is that it cannot directly interact with user workflows. For example, data locality frameworks actively set job input to match the distribution of datasets. Our system instead has to optimize data placement to match the splitting already used by jobs.

3. HTDA Prototype

Our approach is prototypically implemented as the High Throughput Data Analysis middleware [5]. At its core is a generic node application, which is deployed on worker nodes and service machines. The HTDA nodes implement all facilities to join together to a single pool.

Each node runs one or several modules which implement the actual services:

- A *Provider* on each worker node, which adds, maintains and removes local copies of files.

- A *Locator* per submission node, which tracks the files available on worker nodes.

- A *Coordinator* per pool, which decides what files to cache and where to do so.

We have deployed our middleware on a portion of the local KIT CMS *HTCondor* cluster. The HTDA section is composed of 4 worker nodes (see Table 1) running *Provider* nodes. A total of 7 file servers mounted via NFS are used.

Table 1: Test Cluster Worker Node

OS		Scientific Linux 6 (Kernel 2.6.32) or CentOS 7 (Kernel 4.4.2)
CPU	2x	Intel Xeon E5-2650v2 @ 2.66 GHz (à 8 cores, 16 threads)
Memory	8x	8GB RAM
SSD	1x	Samsung SSD 840 PRO 512 GB or
	2x	Samsung SSD 840 EVO 256 GB
Network	1x	Intel X540-T1 (10GigE/RJ45)

3.1. Middleware Performance

Being a prototype, we have implemented the middleware in *Python*. This is motivated by the need for rapid development and ease of maintenance. The prototype makes heavy use of abstraction, both between node and module as well as between modules themselves. The implementation is mature enough for stable deployment and operation.

Experiences in terms of system requirements have been unexpectedly good. The only performance critical component, the *Provider* node, has negligible overhead (see Table 2). Its CPU consumption is linear to the frequency of validating files. For our tests, this was once every 5 minutes. Since files are rarely deleted by users before being discarded by our cache, this could be reduced by at least one magnitude.

Table 2: Module Resource Consumption, according to the *ps* utility

module	CPU	RSS
Provider	3.5%	120 MB
Locator	1.0%	60 MB
Coordinator	14.1%	1 GB

3.2. Request Interception

We have implemented the interception of requests via two means: Job meta-data is intercepted via hooks in the *HTCondor job_router* daemon. Application access to data is intercepted on the worker node by a union file system.

Using hooks in the *job_router* has several advantages over the common method of parsing the job queue. Most importantly, we do not have to repeatedly scan the queue for jobs. The selection and tracking is efficiently handled by *HTCondor* itself. The hooks are automatically called on job submission and removal as well as regularly while it is running. This allows for all our services to be event driven.

Since the hooks connect to the pool, communication can easily be optimized. Our hooks are executed often, but skip several updates after updating a job successfully. This naturally leads to spreading out requests if our system cannot service them fast enough. Additionally, hooks can address any end-point of the pool for load balancing. Finally, we can limit how many jobs are connected to our system by *HTCondor* at any time. We can thus handle an arbitrary number of queued jobs.

The downside is that only one type of hook may be active per job. Using hooks, it is not possible to track one job by multiple systems, e.g. our cache and an opportunistic resource provider. This would require creating an intermediate hook calling the services' hooks.

Intercepting read requests via a union file system has proven ideal for performance. We have used Another Union File System in all our setups. It performs the redirection to cache or storage inside the VFS layer of the kernel. Any overhead from this is too small to measure. It is worth noting that this technique is not production ready on Scientific Linux 6. The combination of its kernel and the available *AUFS* 2 may deadlock. However, we have since switched to CentOS 7 and *AUFS* 4, which works flawlessly.

3.3. Compatibility with Volatile Resources

The HTDA middleware is robust against nodes unexpectedly entering and leaving the pool. Any node may keep on functioning on its own. *Provider* nodes maintain existing files, allowing *Locator* and *Coordinator* nodes to work with their last known state for some time. This makes the system intrinsically suitable for deployment on opportunistic resources.

For better adaptability to such resources, the handling of file meta-data and ownership may be improved in the future. At the moment, we assume reading from remote caches has no benefit over reading from the original source. Thus files and their meta-data are owned by the

Provider maintaining them. To allow shared caches accessed from multiple hosts, it would be better to have both owned by the cache device. An attached *Provider* would take ownership only temporarily. If the opportunistic worker node hosting it shuts down, the data may persist on the shared device and a new *Provider* may take ownership.

4. Conclusion

Data locality is an important approach for scalable data analysis. To integrate data locality into HEP workflows, we have created a new approach to transparently enhance batch systems. This is based on a pool of coordinated caches, providing files used by jobs locally on worker nodes.

There are several advantages intrinsic to our approach, which make it suitable for use in HEP. Since we target the batch system as a consumer, our system must only provide frequently used data. An arbitrary number and volume of data servers may provide infrequently used data. The system is by design transparent to users and existing workflows. It can thus be added seamlessly to existing infrastructure without negative side effects.

We have implemented our system as a prototype and successfully used it for CMS run 2 analyses. Being the first of its kind, there are several features that may be improved or expanded in the future. These mainly concern the applicability to other setups, such as opportunistic resources or shared cache volumes. The system itself is mature enough for active use in dedicated batch systems.

References
[1] Fischer M, Metzlaff C, Kühn E, Giffels M, Quast G, Jung C and Hauth T 2015 *J. Phys.: Conf. Ser.* **664** 092008
[2] Fischer M, Giffels M, Jung C, Kühn E and Quast G 2015 *J. Phys.: Conf. Ser.* **608** 012018
[3] Giffels M, Hauth T, Polgart F and Quast G 2015 *J. Phys.: Conf. Ser.* **664** 022022
[4] The Apache Software Foundation 2015 Apache hadoop URL `https://hadoop.apache.org`
[5] Fischer M, Metzlaff C and Giffels M 2015 HPDA middleware repository URL `https://bitbucket.org/kitcmscomputing/hpda`
[6] Thain D, Tannenbaum T and Livny M 2005 *Concurr. Comput.: Pract. Exper.* **17** 2–4
[7] Paul S and Fei Z 2001 *Comput. Commun.* **24** 256 – 268
[8] Russo S A, Pinamonti M and Cobal M 2014 *J. Phys.: Conf. Ser.* **513** 032080
[9] Lehrack S, Duckeck G and Ebke J 2014 *J. Phys.: Conf. Ser.* **513** 032054
[10] Yang W, Hanushevsky A B, Mount R P and the Atlas Collaboration 2014 *J Phys.: Conf. Ser.* **513** 042035
[11] Blomer J and Fuhrmann T 2010 A fully decentralized file system cache for the cernvm-fs *2010 Proc. of 19th Int. Conf. Comput. Commun. and Netw. (ICCCN)* pp 1–6
[12] Weitzel D, Bockelman B and Swanson D 2015 Distributed caching using the htcondor cached *Proc. for Conf. Parallel and Distrib. Process. Techn. and Appl.*