Provisioning of data locality for HEP analysis workflows

Provisioning of data locality for HEP analysis workflows

C Heidecker, M J Schnepf, E Kuehn, M Fischer, M Giffels and G Quast

KIT - Karlsruhe Institute of Technology, Germany

E-mail: {christoph.heidecker, matthias.schnepf, eileen.kuehn, max.fischer, manuel.giffels, guenter.quast}@kit.edu

Abstract. The heavily increasing amount of data produced by current experiments in high energy particle physics challenge both end users and providers of computing resources. The boosted data rates and the complexity of analyses require huge datasets being processed in short turnaround cycles. Usually, data storages and computing farms are deployed by different providers, which leads to data delocalization and a strong influence of the interconnection transfer rates. The CMS collaboration at KIT has developed a prototype enabling data locality for HEP analysis processing via two concepts. A coordinated and distributed caching approach that reduce the limiting factor of data transfers by joining local high performance devices with large background storages were tested. Thereby, a throughput optimization was reached by selecting and allocating critical data within user work-flows. A highly performant setup using these caching solutions enables fast processing of throughput dependent analysis workflows.

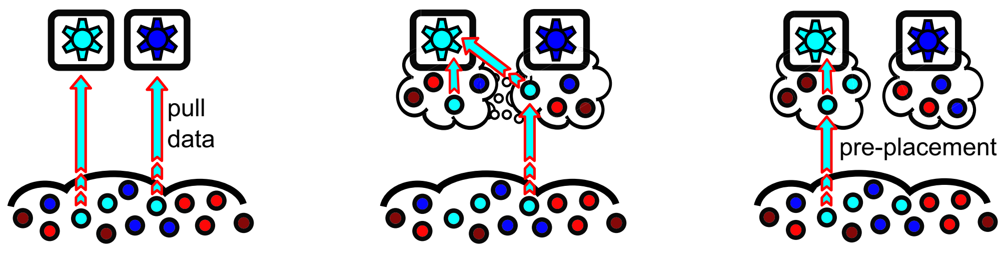
1. Introduction

Performance of data analysis applications is primarily bounded by how fast data is read and processed [1]. While processing speed depends on local resources, read speed may depend on a series of connections to a remote data storage. A close proximity of processing and storage resources can improve bandwidth by avoiding bottlenecks and congestion of shared connections. The term *data locality* refers to this proximity of processing applications and data sources.

For High Energy Physics (HEP), the granularity of the Worldwide LHC Computing Grid (WLCG) [2] allows data locality only at the scale of computing centres. In contrast, distributed storage frameworks [3, 4] achieve data locality at the scale of racks and hosts. However, this approach requires high-performance storage on processing nodes with sufficient volume to hold all data. Given large data volumes and only partial data access at any time, as is common in HEP, the utilisation of such a system is low.

In the past, we developed on-demand data locality at host granularity [1, 5, 6, 7, 8] for HEP analyses: individual, high-performance caches in a batch system are coordinated to provide popular data from background storage. This significantly improves processing speed of recurring analyses, which also frees resources for unique analyses.

However, our research suggests that host granularity for data locality is not always desirable: optimal locality provides excess read speed compared to processing speed, and considerably complicates scheduling. Instead, data locality is best approached as a spectrum (see Figure 1), where performance limits and infrastructure complexity are balanced.



(a) Global access (b) Adjacent access (c) Local access

Figure 1: Spectrum of data locality: Treating data locality as a binary decisions, data consumers access data either globally or locally. However, this also defines the extremes of a spectrum, where data is accessed with a given adjacency, that is a *level of data locality*.

2. Caching for on-demand data locality

Data locality via caching can be split into two separate challenges: First, selecting which input data to provide in order to improve performance. Second, directing consumers to the location of their input data. This translates to two interleaved scheduling problems, both suitable to optimise data processing.

We use a generalised model for data processing, derived from HEP batch processing but not strictly bound to it: Data analysis is organised as *workflows*, each applying a specific analysis application to a *dataset*, a subset of the entire data volume. This is realised by executing multiple *jobs* concurrently, instances of the same application processing non-overlapping portions of the dataset. The execution environment is distributed, with distinct execution hosts sharing a single global namespace to separate data storage. Notably, we do not restrict us to a specific level of detail at which data is read or processed.

2.1. Locality preselection

The processing infrastructure we target is already suitable to efficiently process a fraction of workflows. This is already satisfied if some workflows perform simulation, which require practically no input bandwidth. From our experience, many data processing workflows are also not optimized enough to exhaust bandwidth. In effect, there is a fraction of workflows which does not benefit from caching at all.

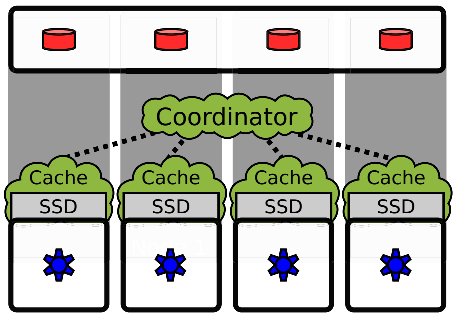
Any data accessed only by such workflows can be culled from caching. This reduces the effective data volume, in turn increasing the relative size of caches. Thus, even with small cache sizes, acceptable cache hit rates and low cache trashing are achievable. While many workflow have no direct benefit of caching, they still benefit from processing resources being freed faster.

2.2. Concurrency optimisation

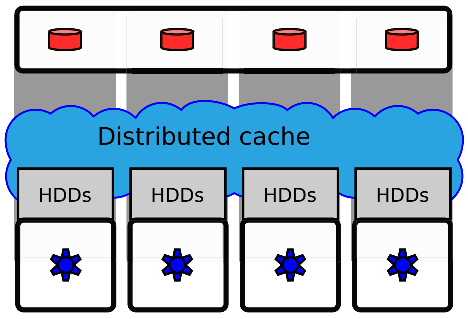
While caches work with individual jobs, our goal is the optimisation of total workflow throughput. This goal must be reflected in the strategy to select and distribute data for caching: the concurrent data access by many jobs of a workflow is the optimisation scenario[1]. Regardless of infrastructure details, this can be effectively achieved by reducing congestion on each data source.

For this goal, individual caches are beneficial as additional, not necessarily faster, data sources. Workflow dataset portions should be spread over as many caches as possible with

[1] Notably, an isolated job has little benefit from caching in our use case. SSD and HDD caches offer bandwidths of 1 Gbit/s to 5 Gbit/s compared to network bandwidths of 1 Gbit/s to 10 Gbit/s.



(a) Coordinated caching



(b) Distributed caching

Figure 2: Caching Scenarios: Coordinated caching uses isolated caches, which form a single service via coordination. Distributed caching forms a single cache, which is delocalised across multiple hosts.

the splitting by jobs. Notably, global storage is also a viable data source; a fraction of each dataset should be excluded from caching to exploit this capacity.

3. Prototypes for HEP batch systems caching

To study the spectrum of on-demand data locality, we have implemented two caching scenarios (see Figure 2): First, caches local to processing hosts are coordinated to implement a global caching strategy. Second, a single cache distributed across all processing hosts of the cluster. For comparability, the prototypes for each scenario are deployed on the same processing hosts.

The prototypes are deployed in the KIT CMS Tier 3 analysis cluster, alongside other extensions [10]. Jobs are scheduled and deployed with the HTCondor batch system [9]: On the one hand, this allows us to inject locality information into job scheduling. On the other hand, services are deployed on a recent operating system, with job dependencies supplied via docker [11]. We operate at a resolution of individual files, since this is well-defined before a job is actually run. For technical details of our middleware, see previous publications [1, 5, 6, 7, 8].

3.1. Coordinated caching

To achieve ideal data locality, each cache is accessible only from its associated processing host. Jobs read data either from the local cache, or the global storage. Only metadata is accessible across hosts. To form a single service working at the scale of entire workflows, an additional coordination layer is required. Data selection is performed at this global level, and portions of the selected data assigned to specific caches.

Coordination allows the highest control, and thus optimisation, of locality. Both data selection and data locality can be dynamically adjusted to match the currently running and queued workflows. Since components exchange only metadata, the system is highly scalable both horizontally and vertically. In addition, processing nodes can be operated outside of dedicated clusters, as long as a local cache is available.

3.2. Distributed caching

To allow easier data locality, each cache is accessible also from adjacent processing hosts. We achieve this with a single, distributed cache pool spanning across all nearby hosts. Jobs read data from an arbitrary host of the pool, or the global storage. The entire cache pool acts as one service, being able to service entire workflows. While there is no distinction between local and adjacent access, data and jobs do not require assignment to specific hosts.

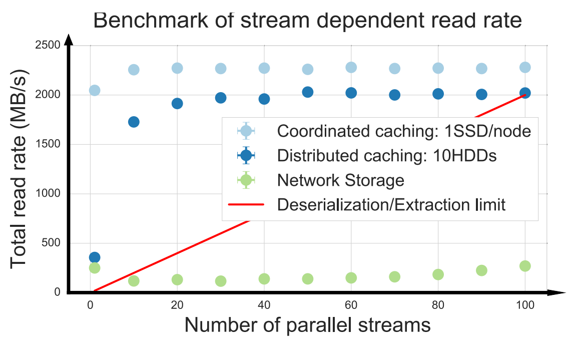


Figure 3: Performance of data sources: Throughput of synthetic, concurrent data streaming. All streams are equally distributed across 4 processing hosts. SSD and HDD devices are local to hosts, while network storage is shared. The Deserialisation/Extraction limit is extrapolated from analysis workflows.

Distribution simplifies job and data management, eliminating the risk of cache misses due to placement. A global strategy can be directly applied to the entire cache pool. As the entire cache pool is used at once, concurrency of data access is automatically smoothed out between hosts; cache devices on each node do not have to deliver a broad performance range. Finally, capacity of resources between hosts can be used, such as fast inter-rack network.

4. Experience and evaluation

We have used processing hosts in a single rack for controlled evaluation of the two caching scenarios. All data accesses use POSIX, with an overlay file system [12] transparently merging cache and background storage. While coordinated caching directly uses SSD devices, distributed caching uses CEPH [13] to integrate HDDs on all hosts. Notably, the HDD data capacity is 20 times the total SSD data capacity, but we use 2.5 times as many HDD devices. For controlled throughput tests, data has been manually pinned to caches.

4.1. Concurrent throughput

We have deployed both synthetic performance tests (see Figure 3) and orchestrated analysis workflows for testing. In general, caches add significant additional bandwidth of 5 Gbit/s per host. SSD caches offer superior response for few concurrent accesses - even with fewer devices, they offer superior performance for any number of streams. However, the HDD cache still provides comparable overall throughput.

For the synthetic test, throughput for both cache setups saturates at 25% cluster utilisation. However, HEP analysis workflows cannot process this throughput, even under ideal conditions - the ROOT [14] dataformat adds overhead for extracting and deseralising data. Even at full cluster utilisation, both cache scenarios offer sufficient throughput for analyses to be compute bound. As a result, caching a fraction of input data is sufficient for optimal performance.

4.2. Applicability and use-cases

We have found both caching scenarios to significantly improve data analysis throughput. Since both scenarios overprovision throughput, they are equally sufficient for our environment. However, their differences in complexity and scalability make them ideal for different use cases in the future.

Coordinated caching benefits from processing hosts having no additional dependencies on infrastructure. Due to the shared-nothing architecture, caches can be volatile, and can be added and removed from a cluster at any time. This allows dynamically deploying isolated processing hosts outside of dedicated computing centers [15].

Distributed caching offers significantly lower complexity for configuration. Given the lower placement constraint to achieve locality, scheduling requires less tuning. Overall, we feel that distributed caching is the correct choice for dedicated computing centres.

5. Summary and outlook

Data locality is a requirement to exploit available processing resources of modern hardware. Caching offers an effective way to provide data locality on-demand. We have developed and evaluated two similar but distinct caching scenarios: Coordinated caching shares only the most basic information, making it ideal for dynamic processing resources. Distributed caching enhances many processing hosts at once, allowing for a simple adoption in an existing cluster.

Still, the approach offers potential for future improvements and applications. From a conceptual point of view, the two scenarios can be combined - by coordinating cache pools, efficient data analysis can be enabled across clusters. Furthermore, extending support to grid protocols such as xrootd, caching and thus data analysis could be extended to opportunistic resources as well.

Acknowledgment

The authors wish to thank the German Helmholtz Association, and the Karlsruhe School of Elementary Particle and Astroparticle Physics (KSETA) for supporting and funding the work. The authors acknowledge also support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG).

References

[1] Fischer M, Metzlaff C, Giffels M, Jung C, Kuehn E, Hauth T and Quast G 2015 High performance data analysis via coordinated caches, *CJ. Phys.: Conf. Ser.* **664** 092008
[2] Eck C et al., 2005, LHC computing Grid : Technical Design Report, *Technical Design Report LCG*, https://cds.cern.ch/record/840543
[3] The Hadoop project homepage, **URL** http://hadoop.apache.org/
[4] Dean J and Ghemawat S 2004 MapReduce: Simplified Data Processing on Large Clusters, *OSDI'04: Sixth Symposium on Operating System Design and Implementation*
[5] Fischer M, Giffels M, Jung C, Kuehn E, and Quast G 2015 Tier 3 batch system data locality via managed caches, *J. Phys.: Conf. Ser.* **608** 012018
[6] Fischer M and Kuehn E 2016 Data Locality via Coordinated Caching for Distributed Processing, *CLOUD COMPUTING 2016*
[7] Fischer M, 2016, Coordinated caching for high performance calibration using $Z \to \mu\mu$ events of the CMS experiment *Karlsruher Institut für Technologie (KIT)* http://dx.doi.org/10.5445/IR/1000067354
[8] Heidecker C 2017 C Absolute jet energy scale determination at the CMS detector for LHC run 2 at $\sqrt{s} = 13\,\text{TeV}$ using an optimized processing setup, URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48877 [Accessed 2017-10-12]
[9] Thain D, Tannenbaum T and Livny M 2005 Distributed computing in practice: the Condor experience, *Concurrency Computat.: Pract. Exper.* **17** 32356 (doi: 10.1002/cpe.938)
[10] Fischer F et al. 2017 On-demand provisioning of HEP compute resources on cloud sites and shared HPC centers, *Journal of Physics: Conference Series*
[11] Schnepf M 2017 Calculation of cross-section limits for the production of single top quarks in association with a Higgs boson using container technologies, URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48876 [Accessed 2017-10-12]
[12] Junjiro Okajima, "aufs" [software], version 4.13.5, Available from http://aufs.sourceforge.net/ [accessed 2017-10-16]
[13] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn, 2006, "Ceph: A Scalable, High-Performance Distributed File System", *USENIX Association* Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI 06), http://ceph.com/wp-content/uploads/2016/08/weil-ceph-osdi06.pdf
[14] CERN, "ROOT - Data Analysis Framework" [software], version 5.34, Available from https://root.cern.ch/ [accessed 2017-10-16]
[15] Schnepf MJ et al. 2017 Mastering Opportunistic Computing Resources for HEP, *Journal of Physics: Conference Series* Proceedings of 18th International Workshop on Advanced Computing and Analysis Techniques (to be published)