# Data Locality via Coordinated Caching for Distributed Processing

Max Fischer and Eileen Kuehn
Karlsruhe Institute of Technology
76344 Karlsruhe, Germany
Email: {max.fischer, eileen.kuehn}@kit.edu

***Abstract*—Modern data analysis methods often rely on data locality. Processing applications are executed directly where data is stored. Frameworks enabling this require their own specialised environment and modifications. We propose an alternative approach using coordinated caching integrated into classic batch systems. A custom middleware layer provides relevant data locally on worker nodes. Most importantly no modifications are needed to add data locality to existing workflows. However, considerably more factors must be addressed by distributed caches compared to isolated ones. We investigated our approach with theoretic modelling, simulations, and a prototype implementation. Our evaluations show promising results for both applicability and performance.**

***Keywords*–Cooperative caching; Coordinated caching; Distributed caching; Batch production systems; Distributed processing.**

## I. INTRODUCTION

Caching as an enabler for data locality is an important topic for distributed data processing. As workflows usually process only a fraction of data frequently. It is thus inefficient to provide all data locally. In addition, changing workflow requirements in batch systems require a dynamic but coordinated caching approach.

Motivated by this, our approach enables data locality for batch systems with minimal requirements. The basis for our efforts are batch clusters which read data via network from dedicated fileservers. To make remote data available locally, a series of caches on worker nodes can be used. We propose a coordination layer that combines individual caches into a single pool.

Our approach stands out from existing caches by its scope and subject: for one, individual caches work on the scale of single machines. We target the entire batch system as a single entity. Also, distributed caches commonly target resource providers, e.g. web caches. In contrast, our approach targets the batch system as a resource consumer.

In this paper, we exemplarily consider the High Energy Physics (HEP) data analysis workflows of groups at the Karlsruhe Institute of Technology [1]. In general, HEP experiments of the Large Hadron Collider [2] are amongst the largest producers of scientific data in terms of data volume. Handling this data is performed in iterative workflows at different scopes. This ranges from reconstruction of raw data at a global scope [3] to the analysis of subsets of data at the scope of university groups.

In theory, HEP analyses conform to principles of data locality based processing. Analysis workflows execute multiple instances of an analysis application in a distributed environment. Each instance extracts the same set of variables from its share of an input data set. All sets of extracted variables are then merged. This corresponds to the map reduce method employed by frameworks such as Hadoop.

However, the wide range of HEP workflows and number of collaborating scientists dictate the use of established applications and frameworks. This also implies constraints, which are not satisfied by modern analysis frameworks. For example, HEP data is commonly stored in the ROOT binary file format. This format cannot be easily split or read from a stream, as is common in the Hadoop framework. The largest volume of data is used infrequently for validation and crosschecking.

We have performed modelling (Section II) and simulations of the situation (Section III). The estimates on architecture and scale are motivated by our own working experiences with the HEP analysis groups. From this, we conclude that classic batch processing can be considerably improved with cache based data locality. Our approach for data locality is based on a concept for coordinated caching (Section IV). To test our conclusions, we implemented a prototype of a caching middleware for batch systems (Section V). First experiences show promising improvements regarding performance and throughput (Section VI).

### A. *Related Work*

Many individual sub-topics of our work have been focus of past research. In general, the approaches show considerable advantages over naive caching. However, no existing work matches the scope and applicability that is needed for HEP workflows.

Distributed caching has been studied extensively. For example, simulations on distributed caching with centralised control [4] show considerable improvements in hit ratio and throughput compared to independent caches. However, research usually focuses on the perspective of data providers, not consumers. Usage metadata is thus not taken into account, limiting the granularity required for data locality.

The CernVM File System (CVMFS) uses independent caches for software provisioning [5]. It is used to make shared software frameworks available in grid and cloud environments. The prototypical CacheD service of the HTCondor batch system caches binaries executed by jobs [6]. This minimises traffic, considerably speeding up deployment of multiple jobs on low-throughput networks. Both of these approaches show that caching is beneficial for batch processing. It improves throughput and allows deployment on resources without high throughput network. However, these approaches target only items which are the same for all jobs.

User communities have made attempts to introduce data locality and/or caching to their workflows. On the one hand, the ATLAS collaboration demonstrated analysis speedup by using a central SSD cache for network attached HDD storage [7]. In this setup, data locality is improved but the cache is still accessed remotely.

On the other hand, attempts have been made to deploy HEP workflows on Hadoop [8], [9]. This provides advanced performance and scalability. However, extensive compromises have to be made. For example, applications have to be adapted to the I/O model and jobs may access only single files. The automatic distribution of data by the infrastructure is severely limited to ensure data integrity.

## II. Modelling Distributed Caching

The modelling of distributed caches involves more aspects than using local caches on a single host. It therefore increases the complexity. Though the data to cache remain the same, factors such as location, replication, or even splitting, and grouping need to be considered.

We model caching statistically to estimate the probability of cache hits, following *cache hit rate*. Using a factorised approach, we describe issues individually and combine them to a single probability. This allows to assess a detailed cost of naively applying caching to a distributed system.

As shown in Figure 1, a distributed cache can be fundamentally viewed as a single entity. This is an ideal case, where a fraction of the global data volume is replicated on the cache volume. However, each worker node (WN) is actually limited to its local scope. In addition, jobs in a batch system are scheduled without knowledge about cache content. In the worst case, data and job placement may be perpendicular.

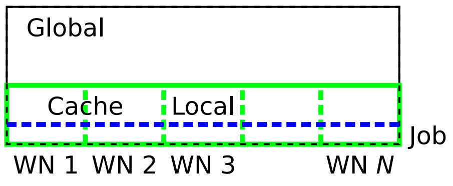

Figure 1. Scopes of distributed caching

To simplify the model, we make two assumptions: First, the volume of individual caches is large compared to items that are cached. Second, the volume of all items is large compared to both the individual and total cache volume. This allows us to treat volumes as continuous, avoiding quantisation and fringe effects. These assumptions mainly simplify the mathematical description. The general conclusions are not changed by this.

### A. A Priori Hit Rate

As with regular caches, we employ a general probability that an item is cached. The choice of this is arbitrary and mainly serves for expressiveness. Without loss of generality, we assume a naive caching approach: the cache is filled with an equal fraction of all data. Thus the probability of an item being cached $P^{\text{base}}_{\text{cache}}$ is the fraction of cache volume $V_{\text{cache}}$ and data volume $V_{\text{data}}$.

$$P^{\text{base}}_{\text{cache}} \propto \frac{V_{\text{cache}}}{V_{\text{data}}} \tag{1}$$

### B. Item Locality

A distributed cache has no single cache volume but actually several separate ones. Cached items can only be accessed efficiently on the host they are located on. We therefore introduce a second probability for item locality, namely that an item is cached on the host it is accessed from, the *local hit rate* $P^{\text{item}}_{\text{local}}$.

If the batch system is not aware of the contents of individual caches, job scheduling is random in relation to item placement. The probability of executing on the correct host is inversely proportional to the number of hosts $N_{\text{hosts}}$. In contrast, creating a number of replicas $N_{\text{replica}}$ on multiple hosts automatically increases the local hit rate. However, replicas reduce the effective total cache volume and thus the overall cache hit rate. Both hit rates combine to the expected hit rate $P^{\text{item}}_{\text{expect}}$.

$$P^{\text{item}}_{\text{local}} \propto \frac{N_{\text{replica}}}{N_{\text{hosts}}} \qquad N_{\text{replica}} \leq N_{\text{hosts}} \tag{2a}$$

$$P^{\text{item}}_{\text{cache}} \propto \frac{1}{N_{\text{replica}}} \tag{2b}$$

$$P^{\text{item}}_{\text{expect}} = P^{\text{item}}_{\text{cache}} \cdot P^{\text{item}}_{\text{local}} \propto \frac{1}{N_{\text{hosts}}} \tag{2c}$$

For naive caching of individual items and no aligned scheduling, we thus expect neither positive nor negative effect from replication. There is no naive approach to mitigate the penalty from distribution over multiple hosts.

In contrast, if we actively align job scheduling and cache locality, the local hit rate becomes a constant – the effectiveness of scheduling. In this case, the positive effect of replication is reduced and ideally eliminated with perfect scheduling. The negative effect of reducing the effective cache volume remains, however. Therefore, the penalty from distributed execution can be mitigated by active scheduling and avoiding replication.

### C. Item Grouping

For efficiency, it is common for any single batch job to process groups of files. In classic batch systems, this grouping is done externally on job submission.

The benefit for each job is proportional to the number of processed items cached locally. We can express the expected efficiency $E^{\text{job}}_{\text{local}}$ as the fraction of expected local files $\left\langle N^{\text{local}}_{\text{items}} \right\rangle$ to total processed files $N_{\text{items}}$. This resolves directly to the local hit rate without aligned scheduling.

$$E^{\text{job}}_{\text{local}} = \left\langle P^{\text{job}}_{\text{local}} \right\rangle \propto \frac{\left\langle N^{\text{local}}_{\text{items}} \right\rangle}{N_{\text{items}}} = \frac{1}{N_{\text{items}}} \sum_{\text{items}} P^{\text{item}}_{\text{local}} \tag{3a}$$

$$= P^{\text{item}}_{\text{local}} \tag{3b}$$

For naive caching with unrelated items, this result is trivial. There are implications if aligned job scheduling is attempted, however. Even when directing jobs to cached items, the fraction of locally available items limits the achievable efficiency. In this context, aligned job scheduling and replication are only useful if cache content is aligned as well.

### D. Access Concurrency

For the scope of our work, a common caching assumption does not hold true: cache access is *not* always superior to remote access. For example, modern SSDs provide throughput at the same order as a $10\,\mathrm{Gbit/s}$ network. The difference is concurrency of accesses: caches are accessed only by local processes, whereas remote data sources can be accessed by all processes.

Modelling accesses for a distributed system in general is beyond the scope of this paper. However, the fact that both local and shared data sources have limited throughput allow for some basic statements. Ideally, both local and shared resources are used to their limit. This automatically implies that perfect cache hit rates are not desirable: a fraction of accesses should always make use of the shared resources available.

## III. Estimates and Simulation

To estimate the benefit of coordinated caching, we have simulated multiple circumstances. The scope and estimates correspond to that of our associated HEP analysis groups.

We model the workflow as a set of application instances, each reading the same amount of data. We have benchmarked common HEP analysis applications to assess realistic limitations. The measurements show a maximum throughput of $20\,\mathrm{MB/s}$ for each instance. This is a technical limit induced by the software frameworks and data formats in use. The total input volume is estimated as $640\,\mathrm{GB}$. This corresponds to the volume of a 2 months data taking period.

For the processing environment, we model our current analysis infrastructure: a set of worker nodes connected to a set of fileservers via a dedicated network, as shown in Figure 2. We assume infinite bandwidth for the fileservers, but model the shared network with its limited throughput of $10\,\mathrm{Gbit/s}$. For processing, we model a number of worker nodes of equal configuration: 32 execution slots, a $4\,\mathrm{Gbit/s}$ SSD cache and a $10\,\mathrm{Gbit/s}$ network connection to the fileservers. For simplicity, we assume that as many instances as execution slots are deployed.

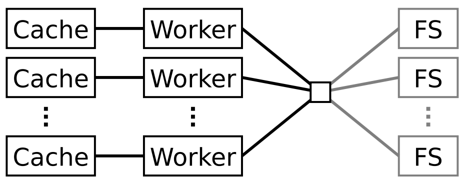

Figure 2. Simulated infrastructure

The results of the simulation can be seen in Figure 3. To rate performance, we have chosen the total processing time. For this observable, lower values are desirable. The cache hit rate to the local cache is used as a free parameter in the simulation. This is motivated by assuming adequate selection of cache content by existing algorithms. Thus, the hit rate is only determined by the scheduling of data and jobs. Dashed, vertical lines indicate the expected local hit rate without coordination.

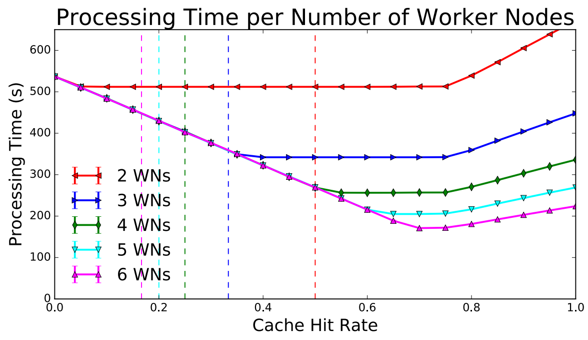

Figure 3. Simulation of workflow runtime

There are two major results from our simulations (see Figure 3): on the one hand, caching enables to scale the infrastructure horizontally. This is especially important since remote access is a bottleneck even with few worker nodes. On the other hand, perfect cache hit rate is actually not desirable. Instead, there is a range of cache hit rates that give best performance. This range is defined by the points where either local or remote throughput is maximized.

Notably, even for perfect caching the expected local hit rate $P^{\mathrm{item}}_{\mathrm{expect}}$ is in this range only for the smallest setup. Therefore, we must coordinate caches to benefit from data locality.

## IV. Concept for Coordinated Caching

Our approach for coordinated caching distinguishes between local and global view as known from batch systems. The local view refers to the individual worker nodes where batch jobs are running. The global view is given by the managing layer where queuing and scheduling of batch jobs is performed.

Based on this scheme, the functionality of a cache can be split into three layers, illustrated in Figure 4:

- the data provisioning on local scale,
- the data access on global as well as local scale, and
- the caching algorithm and distribution logic on global scale.

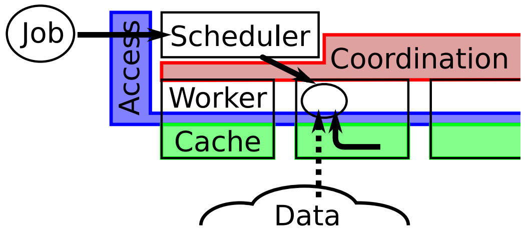

Figure 4. Layers of the distributed cache concept

### A. Provisioning Layer

The provisioning layer handles the actual data on worker nodes. The worker nodes are the only elements with guaranteed access to remote source data. They are therefore the logical place to retrieve and store both actual data and its metadata.

The provisioning layer implements the data provisioning associated with caches: remote items are copied to local cache devices, periodically validated, and potentially removed as needed. In addition, metadata of items, such as size and creation time, can only be collected on the worker node.

Since cache content is by design volatile, the layer also handles content metadata. This ensures that content metadata is as volatile as the actual content: the provisioning layer is the first to notice changes and invalidation of items. In the context of opportunistic resources, worker nodes shutting down neatly remove their allocation information as well.

Every worker node of the provisioning layer is limited to its local scope. Thus, major decisions are outsourced to the global scale. Every worker node exposes information required for cache decisions. The information are composed of its own metadata, such as size of caches, and metadata of items. In return, worker nodes rely on being notified of new items to cache and their relevance.

### B. *Coordination Layer*

The coordination layer is responsible for decision making. Since it is not co-located with any active component, there are no restrictions on resource usage. It can thus fetch, aggregate, and process metadata without impacting other components.

The most important task of the layer is the selection of items to cache. This is fundamentally the same as in other caches. For example, scores can be calculated from access times. Given that arbitrary processing power can be provided, algorithms may be more complex than common. Both metadata and processing resources can be extended as needed.

The additional responsibility of the coordination layer is item placement. This means the deliberate assignment of items to elements of the provision layer. Common features of distributed systems, such as load balancing, must be taken into account. In addition, item relations must be considered for this. For example, common input groups should be assigned together.

### C. *Access Layer*

The benefit of caches depends on low overhead for accessing items. Obviously, executed applications require local access to cached items. On the global scale, the batch system must receive information on item location.

Implementing access to cached items for executed applications on worker nodes depends on the actual setup of batch jobs. Primarily, the access protocol is the deciding factor. In principle, local redirection to cached items is sufficient.

Similarly, details of integrating with a batch system are setup specific. In any case, information on item location must be provided to the batch system. This is the responsibility of thin front ends, which forward information from coordination and cache layer. Functionally, this is similar to lookup services in distributed file systems, e.g. the NameNode of Hadoop FS. However, since all information originates in coordination and cache layer, our front ends are expendable.

## V. HTDA Middleware Prototype

We have implemented a prototype of the coordinated caching concept: the High Throughput Data Analysis (HTDA) middleware [10]. The test environment and community are the HEP analysis groups of our university. We have deliberately kept dependencies on HEP software small, however. The prototype supports the HTCondor batch system [11] and applications performing POSIX I/O.

Architecturally, we build on a general purpose framework for a pool of worker nodes. Every instance of our application represents an individual node. The nodes are loosely coupled. They form a pool using stateless communication. Every node hosts component plugins, which collaborate in the abstracted pool. The component plugins implement the actual layers described in the previous chapter.

We currently use three node types that directly correspond to each layer:

- Provisioning nodes represent the provisioning layer. They stage, maintain, and validate cached files on worker nodes.
- Coordinator nodes represent the coordination layer. They aggregate metadata, and select and assign files for caching.
- Locator nodes provide locality information for the access layer. They act as proxies to the provisioning nodes.

In addition, we enforce access with hooks in the batch system and a redirection layer in each worker node's virtual file system.

### A. *Coordination of Provisioning Nodes*

Coordination of provisioning nodes is based on the scoring mechanism of files. We use a 1-dimensional score to express the relevance for caching. The score is simple to transmit, unambiguous to interpret, and simplifies many algorithms with clear break conditions.

The file score caries an implicit command: files rated higher are assigned with higher priority to provisioning nodes. If space is required for new files, existing files with low scores are discarded first. However, relations between files are not apparent from the score. They are purely handled at the coordination level.

The score of files is calculated inside the coordinator node. A factorised approach is used to express multiple perspectives. Usage prediction is performed using a Least Recently/Frequently Used (LRFU) algorithm [12]. The algorithm is staged as a plugin. Other scoring algorithms can be used if needed. Additionally, we exemplarily model adequacy for caching: exceptionally small and large files are penalised to ease allocation and overhead.

Assignment of files to provisioning nodes is performed separately. Files accessed by the same job are grouped together. These groups are spread evenly amongst all provisioning nodes. Since files may belong to multiple groups, approaches such as distributed hash tables are not adequate. Instead, we exploit that grouping is deterministic and certain groupings are more likely to occur. This allows us to use a heuristic approach, which has proven itself adequate for our target community.

Groups scored highest are allocated first. If any files of a group are already allocated, the node with the highest file count is chosen. Otherwise, a node is chosen at random. This is performed until all files are allocated or no free space is left. Finally, the allocated files and their scores are pushed to each provisioning node individually.

Provisioning nodes only operate on the limited set of assigned files. Since they perform the actual data access, they are the final authority for staging files. This requires limited autonomy for internal allocation and rearranging of files. Thus provisioning nodes expose their allocation to the coordination layer. This allows to iteratively adjust local allocation and global distribution.

### B. Application Access

A major motivation for our development is to preserve existing workflows. Therefore, all accesses must be implemented in a transparent way. To be generally applicable, we must also avoid dependencies on HEP specific software.

The HTCondor batch system natively allows to insert hooks into batch job handling. On batch job submission and finalisation, various job metadata is extracted: this includes input files, resource usage, owner and workflow identifiers. During handling of the batch job, job requirements are updated to prefer hosts caching required input files. This entire process is completely transparent to users.

We exploit HTCondor's rank based scheduling. Users can provide rating functions for worker nodes, e.g. to prefer faster machines. We add our own rating, based on file locality. Thus, locality is preferred in general but can still be overruled by users if hosts are not interchangeable. In addition, we use dynamic requirements to avoid jobs waiting indefinitely for a perfect host. We require worker nodes to satisfy a specific locality rating, which decreases over time.

Application file access on worker nodes is performed using POSIX system calls. Therefore, accesses must be redirected in the virtual file system layer of the operating system. We use a union file system, specifically AUFS [13], to squash cache file systems on top of network file systems. This redirects read access preferably to caches, with write-through to the network storage.

## VI. BENCHMARKING AND EXPERIENCES

We made overall positive experiences with our concept and prototype. Our test deployment is in operation since 6 months. The integration into infrastructure and workflows is fully transparent. Applications subject to caching show notable improvements in runtime.

Our test cluster features four worker nodes (see Table I). Each has 32 logical cores, 512 GB SSD cache and a 10 Gbit/s network interface. A total of 7 fileservers is available on each worker node. The global services for the middleware and batch system are on separate machines.

TABLE I. TEST CLUSTER WORKER NODE

| | | |
|---|---|---|
| OS | | Scientific Linux 6 (Kernel 2.6.32) |
| CPU | 2x | Intel Xeon E5-2650v2 @ 2.66 GHz ( 8 cores, 16 threads) |
| Memory | 8x | 8GB RAM |
| SSD | 1x | Samsung SSD 840 PRO 512 GB or |
| | 2x | Samsung SSD 840 EVO 256 GB |
| HDD | 4x | WDC WD4000 4 TB |
| Network | 1x | Intel X540-T1 (10GigE/RJ45) |

### A. Middleware and Infrastructure

The overhead from our middleware is well acceptable. On worker nodes, the software consumes $(20 \pm 5)\,\%$ of a single CPU. We consider this to be a reliable worst case estimate: first, the frequency of file validation and allocation is very high for testing purposes. This is the majority of actions performed on worker nodes. A factor 2 to 10 less is reasonable for production. Second, our prototype is written in python 2.7 and interpreted with CPython. An optimised implementation in a compiled language is guaranteed to be faster in production. We also investigate other interpreters, e.g. pypy, which provide 3 to 5 times better efficiency in our tests.

The metadata is negligible compared to the actual data. A $500\,\mathrm{GB}$ cache with 7000 cached files has $(3.0 \pm 0.5)\,\mathrm{MB}$ of persistent metadata. The memory footprint of the application is of the same magnitude. The metadata aggregated in the coordination layer is on the order of $250\,\mathrm{MB}$. Communication overhead between nodes is unnoticeable compared to data transfers.

The experience with the access layer using AUFS is mixed. There is virtually no overhead on reading performance compared to direct access. Even during concurrent accesses the full cache device performance is available. However, we have repeatedly observed spontaneous performance degradation and crashes of the union mount service.

We assume the deficiencies to be caused by the old kernel of the Scientific Linux 6 [14] operating system required by our target community. To validate this assumption, we performed tests on the same hardware using CentOS 7 [15] with kernel 4.4.1. The tests revealed none of the deficiencies experienced with Scientific Linux 6. Still, a redirection layer tailored to our access pattern may be more suitable for production deployment.

### B. Integration and Performance

Our prototype operates transparently to users. Applications executed on our worker nodes require no changes. Workflows need to be adapted in one single point: required input files have to be reported explicitly to the batch system to benefit from caching. Since users delegate job submission to tools, we are able to automatise this.

The number of metrics which can be used to assess performance are numerous. A simple and illustrative method is to track individual reference workflows. These are regular end user analyses, extended to collect lightweight performance statistics. Figure 5 shows the distribution of execution times of individual jobs for one workflow. Lower execution times are better. The blue area represents jobs run with the cache disabled. The green area represents jobs run with the cache enabled.

Considerable improvements in execution time indicate appropriate data provisioning and batch job scheduling. Indeed we observe consistently improved execution times when our HTDA middleware is enabled. Highly data driven workflows have been sped up by a factor of 4. Workflows without cached data also benefit from this, as cluster and network utilisation is lower.

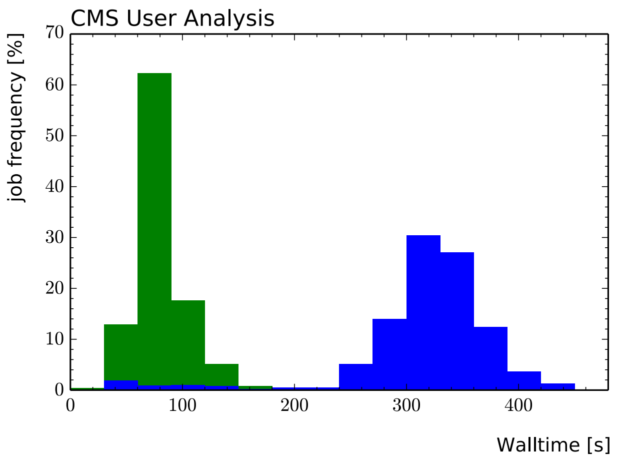

Figure 5. Performance of reference analysis

## VII. Conclusion and Outlook

Modern science is able to collect vast amounts of data. Analysing the increasing volumes is a challenge in itself, however. Solutions exist but are not necessarily applicable. We have therefore investigated an alternative to transparently provide data locality, specifically in batch systems.

We propose a dedicated cache on the scope of an entire batch system. Modelling shows that using individual caches on worker nodes is not efficient. Simulations reveal that high cache hit rates do allow for improved throughput. This is only feasible when actively coordinating individual caches.

Our approach to coordinated caches uses three layers. The provisioning layer is composed of agents on worker nodes, handling data directly. The coordination layer acts on a global scale, coordinating caches based on available metadata. The access layer wraps around workflows on both scales to provide an interface to our system.

To test our estimations and concept, we implemented a prototype of the proposed system – the HTDA middleware. The current implementation handles several aspects of our considerations. This system has already proven to speed up data analysis by a notable factor.

The concept allows room for further research. As our simulations show, perfect hit rate is not desirable. This may be considered in the selection or distribution algorithm to increase effective cache volume. Compared to other cache solutions, our scope is at magnitudes more of processing resources and magnitudes less of turnaround time. Data selection algorithms may therefore be drastically expanded. Furthermore, arbitrary sources for local and external metadata can be considered.

## Acknowledgment

The authors would like to thank all people and institutions involved in the project Large Scale Data Management and Analysis (LSDMA), as well as the German Helmholtz Association, and the Karlsruhe School of Elementary Particle and Astroparticle Physics (KSETA) for supporting and funding the work.

## References

[1] J. Berger, F. Colombo, R. Friese, D. Haitz, T. Hauth, T. Müller, G. Quast, and G. Sieber, “ARTUS - A Framework for Event-based Data Analysis in High Energy Physics,” ArXiv e-prints, Nov. 2015.

[2] The Large Hadron Collider. [Online]. Available: http://home.cern/topics/large-hadron-collider [retrieved: Nov 12, 2015]

[3] The Worldwide LHC Computing Grid. [Online]. Available: http://wlcg-public.web.cern.ch [retrieved: Nov 13, 2015]

[4] S. Paul and Z. Fei, “Distributed caching with centralized control,” Computer Communications, vol. 24, no. 2, 2001, pp. 256 – 268. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366400003224

[5] J. Blomer and T. Fuhrmann, “A fully decentralized file system cache for the cernvm-fs,” in Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on, Aug 2010, pp. 1–6.

[6] D. Weitzel, B. Bockelman, and D. Swanson, “Distributed caching using the htcondor cached,” in Proceedings for Conference on Parallel and Distributed Processing Techniques and Applications, 2015. [Online]. Available: http://stacks.iop.org/1742-6596/513/i=3/a=032054

[7] W. Yang, A. B. Hanushevsky, R. P. Mount, and the Atlas Collaboration, “Using solid state disk array as a cache for lhc atlas data analysis,” Journal of Physics: Conference Series, vol. 513, no. 4, 2014, p. 042035. [Online]. Available: http://stacks.iop.org/1742-6596/513/i=4/a=042035

[8] S. A. Russo, M. Pinamonti, and M. Cobal, “Running a typical root hep analysis on hadoop mapreduce,” Journal of Physics: Conference Series, vol. 513, no. 3, 2014, p. 032080. [Online]. Available: http://stacks.iop.org/1742-6596/513/i=3/a=032080

[9] S. Lehrack, G. Duckeck, and J. Ebke, “Evaluation of apache hadoop for parallel data analysis with root,” Journal of Physics: Conference Series, vol. 513, no. 3, 2014, p. 032054. [Online]. Available: http://stacks.iop.org/1742-6596/513/i=3/a=032054

[10] M. Fischer, C. Metzlaff, and M. Giffels. HPDA middleware repository. [Online]. Available: https://bitbucket.org/kitcmscomputing/hpda [retrieved: Nov 12, 2015]

[11] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: The condor experience,” Concurrency and Computation: Practice and Experience, vol. 17, 2005, pp. 2–4.

[12] D. Lee, J. Choi, J.-H. Kim, S. Noh, S. L. Min, Y. Cho, and C. S. Kim, “LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies,” Computers, IEEE Transactions on, vol. 50, no. 12, Dec 2001, pp. 1352–1361.

[13] J. R. Okajima. AUFS project homepage. [Online]. Available: http://aufs.sourceforge.net [retrieved: Nov 12, 2015]

[14] S. Linux. Scientific linux. [Online]. Available: {https://www.scientificlinux.org} [retrieved: Feb 10, 2016]

[15] C. Project. Centos project. [Online]. Available: {https://www.centos.org} [retrieved: Feb 10, 2016]