EPJ Web of Conferences **337**, 01283 (2025) https://doi.org/10.1051/epjconf/202533701283
CHEP 2024

Reshaping Analysis for Fast Turnaround: Leveraging Concurrency to Reduce Latency in Late-Stage LHC Analysis Workflows

Kevin Lannon[1,*], *Connor* Moore[1], *Barry* Sly-Delgado[2], *Douglas* Thain[2], *Benjamin* Tovar[3], *Austin* Townsend[1], and *Jin* Zhou[2]

[1]Department of Physics and Astronomy, University of Notre Dame, Notre Dame, IN, 46530, USA
[2]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46530, USA
[3]Center for Research Computing, University of Notre Dame, Notre Dame, IN, 46530, USA

Abstract. In the data analysis pipeline for LHC experiments, a key aspect is the step in which particle-level data is reduce to summary statistics allowing insights to be extracted through statistical analysis. Here, we will refer to this step as "analysis." Analysis is a very important part of the pipeline as it is the step where individual researchers exercise their creativity in trying new ideas in the pursuit of discovery. Therefore, a critical metric for the analysis step is turnaround time because it determines how rapidly researchers can explore their space of ideas. We demonstrate our experience reshaping late-stage analysis applications on thousands of nodes with the goal of minimizing turnaround time. It is not enough merely to increase scale: it is necessary to make changes throughout the stack, including storage systems, data management, task scheduling, and application design.

1 Analysis Computing

A central challenge in high energy physics (HEP) is the need to apply computing to the large volume of data generated by experiments in order to extract physics insight. This challenge is currently present at experiments collecting data from proton collisions generated by the CERN LHC, and will only grow more acute as a planned upgrade of the LHC beam intensity—referred to as the High Luminosity LHC or HL-LHC–promises to increase the rate of data produced by LHC experiments by roughly a factor of 10 in the coming years. As there is little chance of increasing the available computational resources supporting LHC data analysis by the same factor, and as we cannot afford to wait a factor of 10 longer to complete analyzing LHC data, this challenge will need to be addressed by exploring alternative approaches to processing data.

Because there is a wide range of computing scope and scale involved in extracting insight from HEP experiments like the CMS [1] or ATLAS [2] experiment at the CERN LHC, it is important that we clearly define what we mean by "analysis" here. A schematic diagram representing the pipeline from proton collision to publication is shown in Fig. 1. The computing pipeline begins by ingesting raw data from the detector and processing it into a format which

*e-mail: klannon@nd.edu

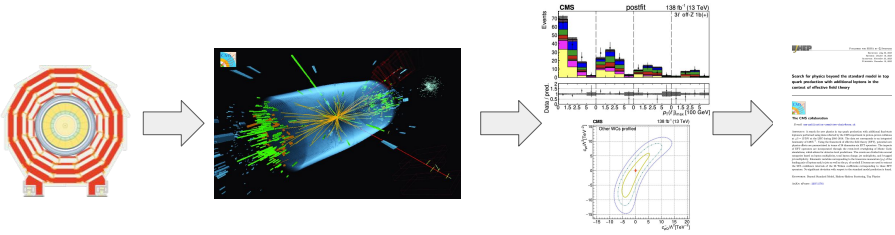

Figure 1. A representation of the computing pipeline from proton collisions to publication for LHC experiments.

Table 1. Summary of *production* versus *analysis* computing for LHC experiments like CMS or ATLAS.

	Production	Analysis
Code	High performance algorithms written in C++	User code written in Python and/or C++
Input	Petabytes	Terabytes
Output	Terabytes	Kilobytes
Resources	Millions of jobs running over weeks or months operated by a small team of experts	Hundreds/thousands of jobs (per user) running for ideally hours by many different, non-expert users

is essentially a list of reconstructed final state particles (electrons, muons, photons, charged and neutral hadrons, etc.). We will refer to this part of the pipeline as *production* Following *production*, the data is further reduced via statistical techniques, like histograms, and eventually parameters of interest and their uncertainties are extracted through statistical analysis. We will call this part of the pipeline *analysis*. Finally, the extracted parameters are used to inform scientific insights expressed in the form of a paper.

Tab. 1 compares the differences between *production* and *analysis* computing. While there are many interesting challenges inherent in *production*, our focus in this paper will be on the challenges of *analysis*. In particular, we will focus on the aspect that goes from processing terabytes of particle level data (e.g. the nanoAOD [3] format from the CMS experiment) into a statistical summary such as histograms. This step involves applying calibrations and corrections to the particle level information, selecting interesting collisions based on the properties of the reconstructed particles, calculating event-level and particle-level quantities of interest, and filling histograms or otherwise extracting summary information through one or more statistical techniques. This is the phase in which many graduate students spend the majority of their scientific effort and creative problem solving; therefore, any improvements in the turn-around time are bound to provide significant boosts in scientific productivity.

2 Reshaping

One way to address the challenge of improving the turnaround time of *analysis* is through an approach called "reshaping." Reshaping refers to increasing processing concurrency in an attempt to reduce the time it takes to complete the processing. Consider Fig. 2. In both plots, the horizontal axis represents the elapsed processing time, while the vertical access shows the amount of concurrency. The plot on the left of Fig. 2 depicts a typical high throughput

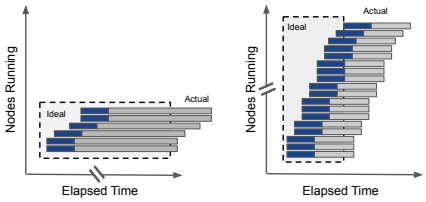

Figure 2. An illustration of the concept of reshaping. The dashed line shows the ideal result while the solid boxes represents a more realistic result with impact from task latency and overhead.

workflow running in parallel on N nodes for t minutes. Assuming more resources are available on which to run the processing, reshaping would transform this application to run on a larger number of nodes, say $10 \times N$ with the hope of achieving significantly shorter elapsed processing times, ideally $t/10$.

The ideal case of reshaping is represented by the dashed rectangles in the figure and can be thought of as a simple rotation of the rectangle in the plane of the plot, preserving the area of the rectangle. However, the latencies and overheads involved in distributing concrete computing tasks to physical resources and getting the processing software running, cause deviations from the ideal picture as illustrated in the figure.

While it is relatively easy to sketch out the concept of reshaping in graphical form, actually achieving it in practical situations requires overcoming two key challenges. First, the computation to be performed needs to be expressed in a flexible enough way that the reshaping transformation can easily be applied. For example, a black-box application that can't be broken down into finer-grained elements cannot easily be reshaped. Secondly, the framework that does the reshaping needs to minimize the latency of dispatching tasks to resources and the overhead of starting the task on the remote node, or these latencies and overheads will completely obscure any benefit that might come from reshaping. We will focus on these challenges as we consider one possible implementation of a software framework cable of reshaping.

3 Analysis Software

As mentioned above, a "black box" application is a challenging candidate for reshaping because the framework has limited insight into the potential amount of concurrency that can be achieved. Unfortunately, it is difficult to go beyond treating an application as a "black box" if the application is allowed to be structured and implemented in a fully general and arbitrary way. To avoid this difficult, we will restrict our consideration only to applications that follow a *columnar* approach. The columnar analysis approach assumes that all analyzed data can be represented in parallel lists (i.e. columns) where the index in the column (i.e. the row) represents data from a corresponding collision, also sometimes referred as an "event." An intuitive representation of a columnar analysis would be a spreadsheet, where each column in the sheet represents a possible quantity that can be measured from a collision, and the rows are used to indicate the values for different collisions. However, we will not limit analyses to such a simplistic columnar approach. In particular, we can allow that columns might be higher dimensional data structure, such that the "entry" in a given row might be a list, a table, or higher dimensional array. In other words, as long as the first index in the column correspond to the "row number" (i.e. corresponding to a specific collision), there can be arbitrarily

Figure 3. Breakdown of the software stack for reshaping columnar data analysis. The left side of the figure represents the software stack components abstractly, while the right side provides specific technology choices for the studies in this paper.

more indices. Furthermore, the data structures for columns do not need to be "rectangular," meaning that the additional indices don't have to be fixed length, but might vary in length from one "row" (i.e. collision) to the next. Data structures with variable length indices are sometimes referred to as "jagged."

In taking a columnar approach, we will assume that the data is laid out in memory (or on disk) such that column values occupy contiguous blocks of memory. This allows for very efficient bulk processing operations on columns. Such data is well suited to an *array programming* approach, such as that taken by the numpy [4] software and awkward [5] that provides a jagged generalization of the numpy approach. For what follows, we will assume that users are employing just such an approach. That is, the users will be writing "numpy-like" array programming statements to express their analysis on columnar data. We will refer to this as the **Application Code**, which will form the top layer in our software stack.

Fig. 3 represents the rest of the software stack under consideration in this paper. The top layer is the application code, as described above. Below that comes the **Task Graph Manager**. This layer takes the application code and extracts a graph representation of the columnar data and the operations performed on it. Note that in general, the nodes in this graph do not represent "black box" entities like applications and files, but rather a more fine-grained description in terms of array operations and the contiguous (in memory) "chunks" to which they're applied. The graph generated by the **Task Graph Manager** is then passed on to the next layer, the **Task and Data Scheduler**, which takes responsibility for distributing the data across the cluster and sending the array operation nodes as tasks to be executed within the distributed cluster. The final layer of the software stack is the software running on the **Cluster Nodes** that enables them to be harnessed remotely to run the tasks.

For this paper, our software stack is concretely implemented with the following open source software components:

Application Code: Users will construct their application using Coffea [6], which is an HEP data analysis framework designed to facilitate columnar data analysis, built on top of awk-

ward [5] and hist [7]. Using this framework, users will write "numpy-style" code expressing their HEP data analysis, extracting statistical summary information (typically in the form of histograms) from collision-level data.

Task Graph Manager: Dask [8] provides the task graph manager functionality. Among other things, Dask provides the ability to generate a computational graph from a sequence of numpy, dask-awkward [9] expressions into a computational graph, with only minimal modifications to preexisting user code designed to run locally.

Task and Data Schedule: The data and computation contained in the graph created by Dask is consumed and transformed into a form that can be scheduled in the cluster by the TaskVine [10] software. TaskVine takes responsibility for ensuring that the appropriate computational environment and software are available to run the code. TaskVine uses a worker-manager paradigm, and workers cache files in the cluster so that they can be reused by subsequent tasks if appropriate. TaskVine workers are also capable of transferring files between workers in the cluster when a file possessed by one worker is needed by another.

Cluster Nodes: TaskVine can talk to a variety of batch or cloud backend interfaces for scheduling the TaskVine workers to run the tasks generated by the graph generated by Dask. For this work, we use the HTCondor [11] batch system.

3.1 Example Application

For the performance optimization studies discussed below, we will use a representative example application known as DV. The DV application code is written using Coffea and Awkward as described above. DV is designed to analyze data from the CMS experiment stored in nanoAOD [3] augmented with additional information about the constituent particles that have been clustered into jets. DV takes these data and calculates the energy correlation functions (ECFs) [12] based on the jet constituent information, and then outputs the ECF values in parquet format to be fed into a subsequent machine learning pipeline.

As far as examples go, the DV application is somewhat atypical. Calculation of the ECFs, including up to 5-point correlations functions, puts this application on the high end in terms of CPU needs. The input to the application consists of data from 20 million collisions, totaling 160 GB in the enhanced nanoAOD format described above, referred to as PFnano. The output includes data from 5.7 million selected collisions, enhanced with calculated values for about 160 ECFs, requiring 7.6 GB of storage. To accomplish this process, the software stake leverages 4,000-6,000 CPU cores, 400 GB of disk, and roughly 2 GB/CPU core of memory in the distributed cluster for about 6-8 hours of elapsed processing time. The DV application is visualized in graph format in Fig. 4

4 Performance Optimization

As illustrated in Fig. 2 a key to good performance is to start many tasks quickly with low overhead. Any overhead or latency in starting up the tasks will degrade the performance from the ideal case and, in principle, severely limit the benefits of high concurrency.

To illustrate the impact of the lower layers of the software stack, we performed an experiment where we compared a two runs of the DV application (using a partial dataset to limit the run time of the application). The only differences between the two runs were at the **Task and Data Scheduling** layer and in the storage technology used for the data. In one experiment, we used the Work Queue [13] software as the scheduling layer, while the other run used TaskVine in "function" mode. Also, in one run the data is stored on spinning disk in

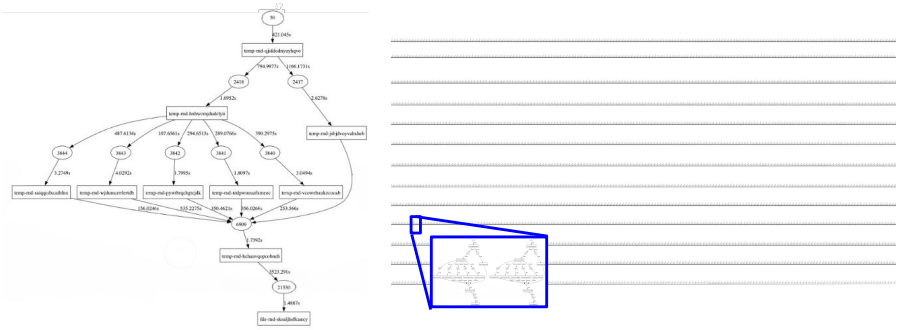

Figure 4. Graph representation of the DV application. On the left, the graph shows the computation for one independent chunk of data. The diagram on the right represents the graph obtained by applying the calculation to a full dataset, which consists of a large number of chunks like the one on the left.

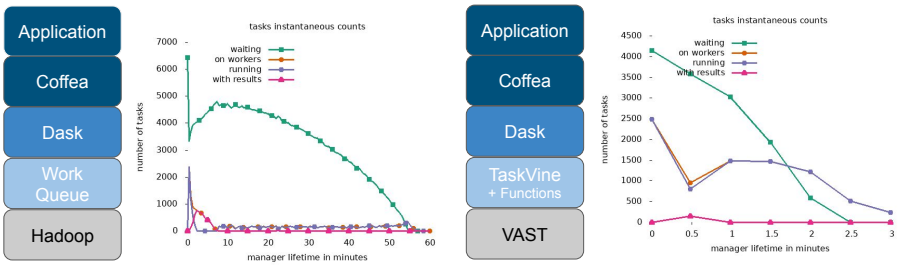

Figure 5. A comparison of two runs of the DV application with the changes in the bottom layers of the software stack highlighted. The second run performs substantially better than the first, even though the top levels of the stack remain unchanged.

storage configured using the Hadoop file system while the second run uses VAST [14] NVMe storage. The difference in overall application performance is represented in Fig. 5. Keep in mind that for these two runs, the **Application Code** layer and **Task Graph Manager** layer are constructed identically between the two examples.

As can be seen in Fig. 5, the elapsed running time for the second run is improved by a factor of approximately 20 compared to the first run. This substantial improvement can be understood as arising from a combination of factors:

- At the **Task and Data Scheduling** layer, TaskVine implements a number of improvements compared to Work Queue, including a more efficient approach to distributing task payload, a more effective approach to caching data in the cluster, and a peer-to-peer mechanism for transferring data between workers in the cluster rather than relying solely on the manager for data distribution.

- The Vast NVMe storage provides significantly better I/O performance, not only in terms of maximum achievable read bandwidth, but also in terms of IOPS.

Although the individual impact of each factor was not measured separately, we tested that using NVMe storage without the improvements in the TaskVine software was not sufficient to produce any noticeable performance improvement.

5 Potential and Challenges

That the performance of the example application was so dramatically improved with techniques applied only at the lowest level in the software stack highlights the potential of this approach and leads to consideration of further possible improvements. A key to this strategy is having a fine-grained graph showing the relation between various parts of the computation and the data necessary to carry them out. The graph description of the calculation makes it possible to examine various aspects of the computation, such as potential concurrency, resource requirements, and data transfer needs, to determine application-specific strategies for improved performance. The graph description also makes it possible to make informed decisions about checkpointing and caching intermediate results from the graph to accelerate repeated runs of computation or make it more resilient to transient node failures. Analyzing the graph may also lead to insights for a particular workflow regarding whether different strategies for evaluating the graph, for example, breadth first versus depth first, would yield better results. Finally, if the graph description is augmented with appropriate hints, the **Task and Data Scheduling** can direct specific parts of the computational graph to the resource that would provide the best performance (for example, GPU versus CPU). Given the range of possibilities, it's clear that the studies in this paper only scratch the surface of what remains to be explored.

While the approach described here offers much potential, there are also challenges to be overcome. Because the task graph breaks the computation into smaller elements than is traditionally present in a "black-box" application approach, there is a need to serialize intermediate production from the computation to be consumed by the next task generated from the graph. This means information which would normally remain stored in memory is transferred between tasks in serialized form using network and storage resources, which can present challenges that application designers may not anticipate, and choices related to serialization, like the trade off between storage space and read/write time posed by compression, raises new optimization challenges. Furthermore, because the user code is transformed into graph representation before execution, with the currently available tools, it is not easy to associate a crash of the running task with a particular line in the source code. This challenge needs to be addressed by developing techniques for maintaining the link between user source code and executing tasks, keeping in mind that even with such a tool, the translation of the source code into graph format may make it difficult for users to understand how certain failure modes arise. Finally, while the hope would be to automate performance optimization, there will always be a need to communicate to users about how the computations in the graph are using resources so that users can identify and resolve any bottlenecks not addressed within the software stack itself.

6 Conclusions

Improving the performance of analysis software is an important part of addressing the data analysis challenges posed by the LHC and in particular, the HL-LHC upgrade. The combination of columnar data analysis software frameworks and graph representations of the resulting

computations offers an interesting new paradigm for tackling that problem. Task graphs offer a rich and flexible way of expressing the computations arising from HEP data analysis. The task graph representation opens the door for various automated optimizations of software performance, such as the reshaping technique discussed in this paper. Reshaping promises to produce dramatic acceleration in computation, but only if the software stack implementing the task graph approach can keep latency and overhead for starting tasks from the graph as low as possible. This is an approach that is ripe for continued exploration, and the examples shared here hopefully represent the beginning of a fruitful line of investigation.

References

[1] S. Chatrchyan, E. de Wolf, P. Van Mechelen et al., The cms experiment at the cern lhc, Journal of instrumentation.-Bristol, 2006, currens **3**, S08004 (2008).

[2] G. Aad et al. (ATLAS), The ATLAS Experiment at the CERN Large Hadron Collider, JINST **3**, S08003 (2008). 10.1088/1748-0221/3/08/S08003

[3] K. Ehatäht (CMS), NANOAOD: a new compact event data format in CMS, EPJ Web Conf. **245**, 06002 (2020). 10.1051/epjconf/202024506002

[4] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith et al., Array programming with NumPy, Nature **585**, 357 (2020). 10.1038/s41586-020-2649-2

[5] J. Pivarski, C. Escott, N. Smith, M. Hedges, M. Proffitt, C. Escott, J. Nandi, J. Rembser, bfis, benkrikler et al., scikit-hep/awkward-array: 0.13.0 (2020), https://doi.org/10.5281/zenodo.3952674

[6] Coffea (2021 [Online]), fermi National Accelerator Laboratory. Available: https://github.com/CoffeaTeam/coffea

[7] H. Schreiner, S. Liu, A. Goel, hist (2023), https://doi.org/10.5281/zenodo.8338901

[8] M. Rocklin, Dask: Parallel Computation with Blocked algorithms and Task Scheduling, in *Proceedings of the 14th Python in Science Conference*, edited by K. Huff, J. Bergstra (2015), pp. 130 – 136

[9] Dask-awkward, https://github.com/dask-contrib/dask-awkward

[10] B. Sly-Delgado, T.S. Phung, C. Thomas, D. Simonetti, A. Hennessee, B. Tovar, D. Thain, TaskVine: Managing In-Cluster Storage for High-Throughput Data Intensive Workflows, in *18th Workshop on Workflows in Support of Large-Scale Science* (2023)

[11] D. Thain, T. Tannenbaum, M. Livny, Distributed Computing in Practice: The Condor Experience, Concurrency and Computation: Practice and Experience **17**, 323 (2005).

[12] A.J. Larkoski, G.P. Salam, J. Thaler, Energy Correlation Functions for Jet Substructure, JHEP **06**, 108 (2013), 1305.0007. 10.1007/JHEP06(2013)108

[13] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, D. Thain, Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications, in *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* (2011)

[14] Vast data, https://www.vastdata.com/