FERMILAB-CONF-18-016-CD

Striped Data Server for Scalable Parallel Data Analysis

Jin Chang, Oliver Gutsche, Igor Mandrichenko, James Pivarski

ivm@fnal.gov

Abstract. A columnar data representation is known to be an efficient way for data storage, specifically in cases when the analysis is often done based only on a small fragment of the available data structures. A data representation like Apache Parquet is a step forward from a columnar representation, which splits data horizontally to allow for easy parallelization of data analysis. Based on the general idea of columnar data storage, working on the [LDRD Project], we have developed a striped data representation, which, we believe, is better suited to the needs of High Energy Physics data analysis. A traditional columnar approach allows for efficient data analysis of complex structures. While keeping all the benefits of columnar data representations, the striped mechanism goes further by enabling easy parallelization of computations without requiring special hardware. We will present an implementation and some performance characteristics of such a data representation mechanism using a distributed no-SQL database or a local file system, unified under the same API and data representation model. The representation is efficient and at the same time simple so that it allows for a common data model and APIs for wide range of underlying storage mechanisms such as distributed no-SQL databases and local file systems. Striped storage adopts Numpy arrays as its basic data representation format, which makes it easy and efficient to use in Python applications. The Striped Data Server is a web service, which allows to hide the server implementation details from the end user, easily exposes data to WAN users, and allows to utilize well known and developed data caching solutions to further increase data access efficiency. We are considering the Striped Data Server as the core of an enterprise scale data analysis platform for High Energy Physics and similar areas of data processing. We have been testing this architecture with a 2TB dataset from a CMS dark matter search and plan to expand it to multiple 100 TB or even PB scale. We will present the striped format, Striped Data Server architecture and performance test results.

1. Introduction

High Energy Physics data analysis is an iterative process of distilling large amounts of data to extract physical information and presenting it in a most optimal way. Typically, the physicist repeats the process of refining the data filtering, data compilation and information representation many times, and reducing the time of individual iteration reduces the time of the overall analysis process and allows the physicist to deliver the results sooner.

Currently, HEP data is stored in files and the analysis is essentially a repeating process of running the analysis software over a large set of files. In order to reduce the iteration time, the physicists reduce the initial dataset (set of files) to a smaller set by using 2 methods:

- Skimming is a process of pre-filtering potentially interesting events so that there are fewer events to process during each run.

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

- Slimming is removing the elements of data structure which are not relevant for the particular analysis the physicist is performing. Typical dataset used for analysis before slimming contains ~1000 columns, whereas typical analysis uses only one or two dozens of these columns.

Once the dataset is reduced, the physicist can run their analysis code over smaller amounts of data in a more efficient way, but data reduction contributes to the overall analysis process and slows it down. In addition, because each physicist or group of physicists have their own way to reduce data, they all tend to replicate data multiple times between all the reduced datasets they create over time.

What we are proposing is a fundamentally different way of storing data (see Fig. 1), which would support a much more efficient and fast data analysis process. We propose to store analysis data in a database with efficient and scalable access and largely eliminate the need to reduce and duplicate data by providing direct and immediate access to the needed data and only to the needed data.

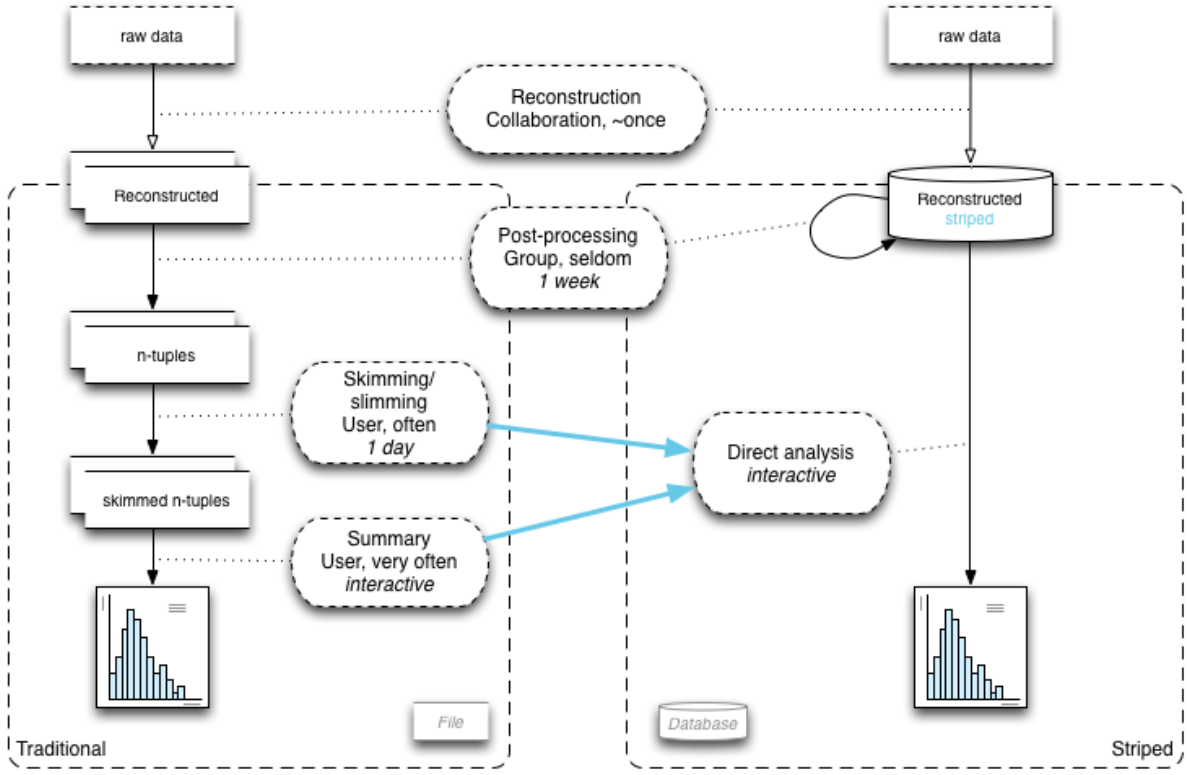

Figure 1. Data analysis process

In addition, we propose to move from the event loop style of analysis to vector-based calculations, which, when combined with functional and/or declarative algorithm representation, can be moved easily from CPU to GPU or other SIMD processing platforms as the underlying computing fabric.

2. Striped Data Representation

High Energy event data structures can be viewed as a recursive list of uniform tuples, some elements of which can be lists of uniform tuples themselves. We developed a format of representing these kind of data structures, based on a columnar approach, suitable to store data in a non-relational database or a BLOB storage. As in columnar representations, the data is represented as a set of columns, each column is essentially a vector of values for a single data attribute for the whole dataset (see Fig. 2). Then we cut columns into segments called “stripes”. Each stripe contains data from a set of HEP events, typically for 1K to 10K or more events. All columns are split along the same event boundaries. For example, if we have columns A and B, we split them in such a way that if we have a stripe for

column A with data for events ranging from 112000 to 112999, then there will be a stripe for column B with data for exactly the same events 112000 to 112999 in the same order. These event ranges are called Event Groups or Row Groups.

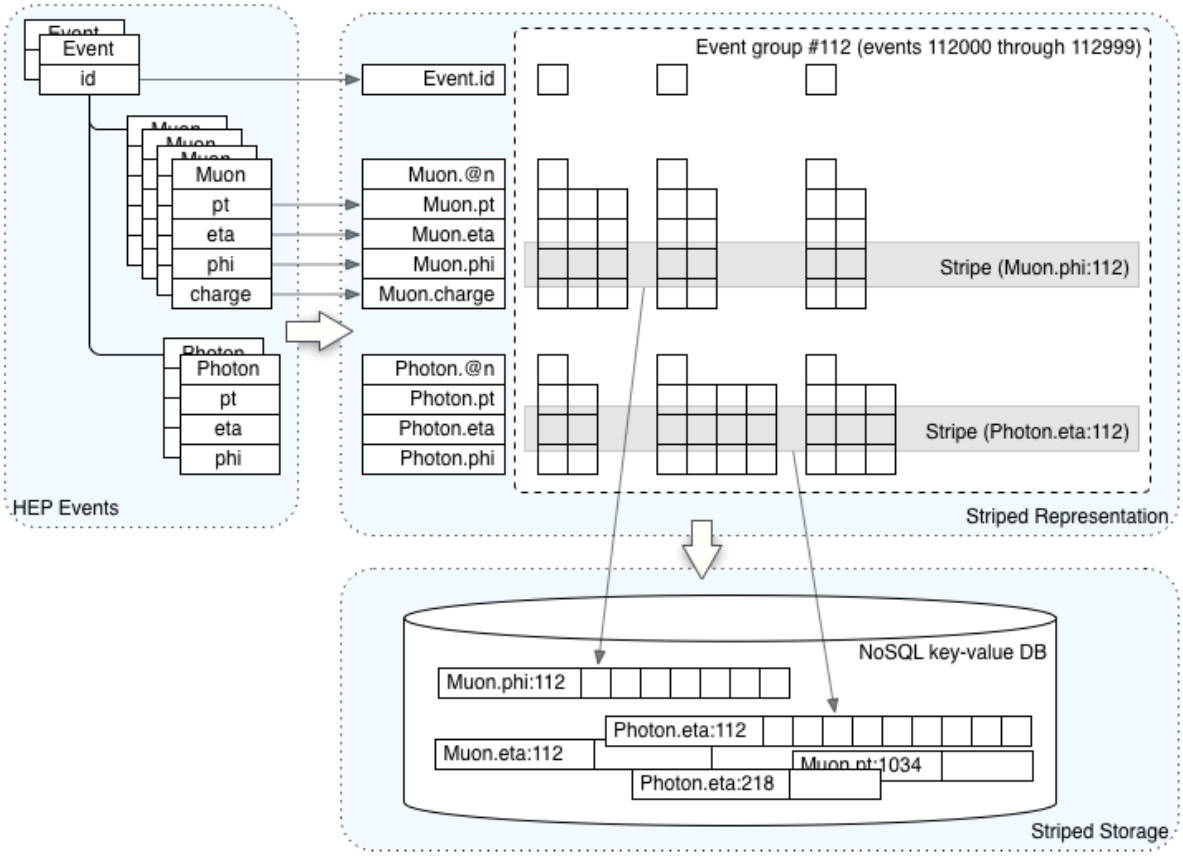

Figure 2. Striped Data Representation

This format of data representation works very well with no-SQL, key-BLOB data storage systems. Each stripe is stored as a BLOB, indexed by the combination of the dataset name, the column name and the row group number. The BLOB contents is in fact a numpy[1] array immediately importable into numpy without any data conversion, which makes the use of such storage very efficient.

Here are the advantages of this data representation for HEP data analysis:

1. We can use pretty much any storage capable of storing key-BLOB pairs. Those include a wide range of no-SQL databases. We can also use relational databases, or even a UNIX file system to store this kind of data and the details of the actual data representation can be hidden behind a uniform data access interface.
2. Columnar representation allows the user to retrieve only columns they need for their analysis and never touch unused columns.
3. Because data is represented in a numpy-friendly way, significant parts of calculations can be performed over stripes as vector operations instead of operations over individual event properties, thus moving large portion of calculations, if not all of them, from the event loop to vector calculations (see Fig. 3).

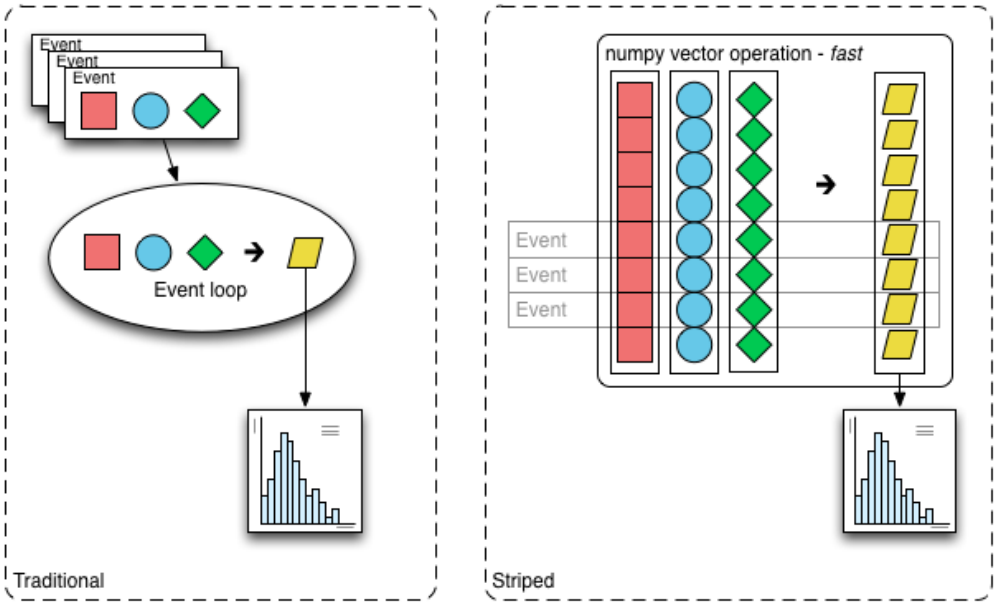

Figure 3. Moving calculations from event loop to vectors

4. Because stripes are almost always immutable (except for rare cases when the dataset needs to be updated in the database, e.g. to correct an error), web services and data caching techniques can be leveraged to significantly increase efficiency of the data management and delivery and reduce development and maintenance cost of the system. This way we can take advantage of the fact that typical data analysis reuses the same data repeatedly multiple times.
5. Using web services as the data delivery mechanism makes it easy to leverage Internet and cloud technologies and provides great deal of flexibility in building distributed data centers, helping members of large international collaborations contribute their resources to the collaboration.

3. Implementation

We implemented a test instance of the striped storage system and a small analysis computing cluster at FNAL (see Fig. 4)

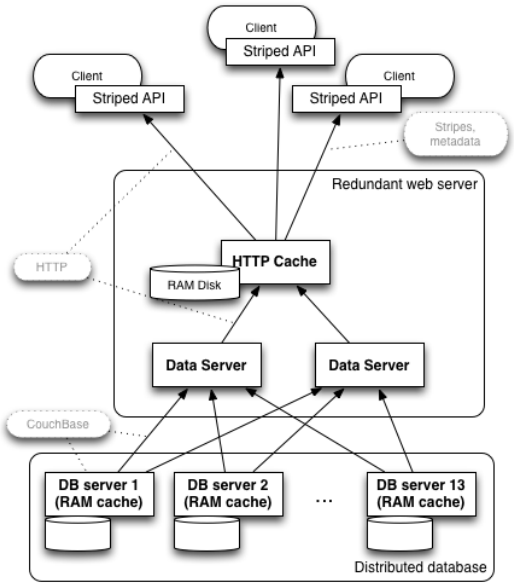

Figure 4. Demo instance of the striped storage

The data storage part was implemented using 13 servers. Each data server was a 8-core computer with 0.9TB data disk. We chose CouchBase[2] as the database software to store data. CouchBase is a rather simple, distributed no-SQL database with memory cache. On top of the CouchBase cluster, we put 2 redundant web servers and then a web cache computer with 20GB RAM-based cache. We used nginx[3] as the web server engine.

We loaded about 2TB of CMS[4] dark matter search n-tuples data into the system.

The computing portion was implemented as a set of independent worker processes in Python, running on 2 16-core computers, so we had 30 workers in total. The data was uniformly distributed among all available workers. Each worker has its own data cache. The algorithm of data distribution among workers always sends the same data to the same worker to make sure they can reuse their local cache. In total, we had 3 layers of RAM cache in the system – the CouchBase own memory cache, web server cache and then the worker local cache.

The analysis platform is also Python based. Currently, it is written in Python 2.7. The user front end is implemented as a iPython/Jupyter notebook with plotting using Matplotlib. The user notebook communicates with the workers via TCP sockets, so the user notebook can run anywhere, as long as it can reach the workers.

On this our cluster, we managed to achieve the performance of about 1 million events per second using only 30 worker processes.

To demonstrate the flexibility of the system architecture, at some point, we created a Docker container with the worker code, and ran the analysis successfully running the workers on the iMac desktop, the analysis 2-node cluster described above and the AWS cloud. All the workers were accessing the data via cached web server located at FNAL.

4. Conclusion

We have developed a format to represent HEP data in a way suitable to be stored in a non-relational database. The data representation format and the database storage mechanism we developed allow to store and analyse data in much more efficient way it is done currently, by eliminating the need to slim and duplicate data. Also storing data in a database with scalable and efficient interfaces allow for scalable and massively parallel data analysis, which largely eliminates the need to skim data. The architecture we propose is highly flexible and does not require co-location of data and computing components. Wide use of Internet and web services technologies allows to deploy the storage as well as the computational components on the cloud. In addition, it allows to have the computing fabric to be distributed over WAN, heterogeneous and dynamic in terms of individual compute node availability. This allows a science collaboration to build distributed computing and storage facilities and have individual members of the collaboration flexibly contribute their resources to the collaboration.

The proposed computational model moves bulk of the computations from the event loop to the vector-based paradigm, which allows to move to functional and/or declarative programming models, which, in turn, allow to move computations easily from CPU to GPU and other SIMD platforms.

Utilization of data caching techniques significantly increases the efficiency of the analysis process, taking advantage of the repeating nature of the HEP analysis process.

We built a demo HEP data analysis platform based on iPython/Jupyter as the user interface.

We were able to build a small 30 cores demo analysis cluster and test its performance on a real 2TB dataset from CMS and achieved a performance of 1 million events per second with all the components written in Python.

The proposed approach has a great potential of replacing existing file-based data analysis models and can significantly increase the efficiency of the analysis process, reduce time to insight, add flexibility in building storage and analysis facilities, and open new areas of HEP research, currently not possible due to a relatively slow analysis process.

5. References

[1] S. van der Walt, S. C. Colbert, G. Varoquaux, The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science Engineering, 2011, Vol. 3

[2] http://couchbase.com

[3] http://nginx.org

[4] Chatrchyan, S. et. al., The CMS experiment at the CERN LHC, JINST, 2008, Vol. 3