Dynamic and on demand data streams

Matteo Duranti^{1,*}, Valerio Formato^{1,**}, and Valerio Vagelli^{2,1,***}

¹INFN Sezione di Perugia, I-06100 Perugia, Italy

²Università degli Studi di Perugia, I-06100 Perugia, Italy

Abstract. Replicability and efficiency of data processing on the same data samples are a major challenge for the analysis of data produced by HEP experiments. High level data analyzed by end-users are typically produced as a subset of the whole experiment data sample to study interesting selection of data (streams). For standard applications, streams may be eventually copied from servers and analyzed on local computing centers or user machine clients. The creation of streams as copy of a subset of the original data results in redundant information stored in filesystems and may be not efficient: if the definition of streams changes, it may force a reprocessing of the low-level files with consequent impact on the data analysis efficiency. We propose an approach based on a database of lookup tables intended for dynamic and on-demand definition of data streams. This enables the end-users, as the data analysis strategy evolves, to explore different definitions of streams with minimal cost in computing resources. We also present a prototype demonstration application of this database for the analysis of the AMS-02 experiment data.

One of the major challenges for the analysis of data produced by high energy physics (HEP) experiments is the efficiency of the data processing on the same data samples. The output of the process of data taking is typically a collection of values produced by the detector electronic channels, that digitizes the interesting information provided by the analog probes. Frequently, the "raw" data are processed using custom reconstruction softwares to produce the "reconstructed, or "reco" files, that contain more advanced information defined on the basis of the raw datas. Raw data are rarely processed multiple times, often only to produce the higher-level data files. Reconstructed data, instead, are typically processed many times and by a larger spectrum of users. For each set a raw data, multiple versions of "reco" data files may be produced depending on advances on the reconstruction software. Data for analysis are stored in Centralised Analysis Facilities (CAFs) for efficient access by end users. CAFs provide the hardware infrastructures and software frameworks for efficient access to files by several users. In order to identify interesting – and typically rare – features in the data it is very common to process the same data samples many times. While end-users prefer to have local access to data on their machines, they are forced to analyze files *in situ* due to the large data set sizes.

To overcome this, a diffuse procedure is that of *data preselection*: if only a subset of data is found to be useful for a particular analysis, that subset is copied into a different file, or "stream". If the data size of streams is much smaller than the original file size, it becomes

*e-mail: matteo.duranti@infn.it

**e-mail: valerio.formato@infn.it

***e-mail: valerio.vagelli@unipg.it

feasible to copy the stream on local machines of end-users. Moreover, usually a simplified version of the original data structure that contains a subset of the data information, called DST, is created by the end user to speed up the data analysis. The data process flow is shown in Figure 1.

Figure 1. Typical process for production of end-user analysis data.

In a typical HEP experiment, data are collected in data acquisition sets referred from here on as *runs*. For each run, a set of *events* is produced. Each event contains the information produced by the experiment, in forms of booleans, indexed and un-indexed discrete values and continuous values x_i . Data are usually analyzed many times in search for hidden correlation between the data variables. A typical data analysis approach used in HEP experiments is the so-called *Tag & Probe* method. A subset of the whole data sample that fulfills user-defined criteria is extracted and analyzed. In the simplest scenario, any single criteria used in the data selection can be mathematically expressed as:

$$g_i(x_1, \dots x_N) = 0 \tag{1}$$

The complete selection is itself a function F , of all g_i :

$$F(g_1(x_1, \dots x_N), \dots g_N(x_1, \dots x_N)) = 0. \tag{2}$$

A *stream* is a subset of events that fulfils any requirement equivalent to Eq 2, and it can be uniquely identified by such condition. A common approach to access to data streams is to clone the selected events and save them in a different file maintaining the same data structure. The stream files, if small enough, can be therefore stored and analyzed locally.

The typical process for production of data streams is sketched in Figure 2. This approach,

Figure 2. Production of data streams.

however, presents some drawbacks. The creation of streams as a copy of a subset of the

original data results in redundant information stored in the CAF filesystems. Moreover, if the definition of stream changes, it may force a reprocessing of the files, with consequent impact on the data analysis efficiency.

We propose a solution for implementation of an analysis framework intended for efficient replicated access to data sets as solution to the drawbacks discussed earlier.

1 The PreselectionDB approach

The solution suggested here is based on the implementation of a database, here referred as PRESELECTIONDB, to store the information used to build the file streams and to use it *dynamically* during the data analysis.

The easiest solution is structured in a two step procedure:

1. creation of the database, an operation to be done only once and consisting in:
 - access to reconstructed data information and test all pre-defined selection criteria;
 - store the information on a database indexed by run and event number;
 - index the database with the data ntuples, for instance using the *run* and *event* numbers;
2. evaluation of the database, an operation that has to be done during each data processing
 - inspect the database for events that fulfill the selection criteria;
 - interesting events are selected dynamically and the database provides the user with a unique identifier of the selected event;
 - the user retrieves on demand from file only the selected events. Non interesting events are not accessed from filesystem at all.

While the creation of the database requires a custom software designed to read and process the original data, the software to access the database can be more flexible and easily interfaced to most of analysis routines.

During the PRESELECTIONDB creation, for each event j^{th} all the criteria g_i , ($i = 1, \dots, n$), in the form of Eq.1 are evaluated. Each evaluation results in a true/false statement (corresponding to the statement “the j^{th} event has / has not fulfilled the i^{th} criterion”), that is stored in the database. Each database entry corresponds to an event and a set of true/false statements.

During the analysis phase, for each event j^{th} a subset of criteria F (and, therefore, a subset of streams) in the form of Eq.2 are evaluated by simply accessing to the database and checking the true/false statements. On the basis of rules defined by the end-user, the event is then further analyzed or skipped. The creation of the database prevents the creation of additional file streams avoiding data duplication. This approach, in fact, uses the database as an ancillary tool that is accessed during the data analysis to dynamically retrieve the events that belong to a particular stream. The concept of usage of the database is sketched in Figure 3.

The use of a database and the dynamical event selection allow to save storage space used to clone the information in the streams. If only the high-level information used to define the stream definitions (the true/false statements of Eq.1 used in Eq.2) is stored, the database will occupy only a fraction of the filesystem size occupied by the whole data. In addition to this, the data and the database could be both resident in an external server, and the users can decide which event to access by querying the database. The structure of the events may be more complex than what described here. An event could have a nested structure, a dynamic

Figure 3. Access to data using the preselection database.

number of objects (i.e. the amount of information changes for each event¹). Anyway, the basic idea holds and can be straightforwardly extended to any data structure.

Further information can be stored in the database in form of *indexing variables*. If the data can be partitioned as a disjoint union of subsets, each unambiguously identified by a (integer) indexing number, it may be useful to store such number for each event. In the case the indexing variable is a continuous variable, it could be useful to store the variable as it is. We suggest, however, a possible solution to store multiple indexing variables loosing minimum information and maximizing the optimization of the data size. Let's assume the continuous indexing variables (χ_i) are single-precision floating point numbers that exist in a domain $\mathbb{D}_i = [\chi_i^{(min)}, \chi_i^{(max)}]^2$. A single-precision floating point number would occupy a 4-byte word according to the IEEE 754 standard if stored plain in the database. However, the domain can be partitioned in n disjoint subsets $\mathbb{S}_i$ such that $\bigcap_i \mathbb{S}_i = \emptyset$ and $\bigcup_i \mathbb{S}_i = \mathbb{D}_i$, an operation that in the physics analysis jargon is usually called "binning". Each subset $\mathbb{S}_i$ represents a unique range that could contain the variable. The index of the subset $\mathbb{S}_i$ containing the value of the variables could be stored in the database. In this procedure, the value of the variable is lost but the index of the subset $\mathbb{S}_i$ can be later retrieved during the analysis phase to access to the range of the variable value. The gain in terms of data storage is evident: the index of the subset $\mathbb{S}_i$ in a partition of $n = 2^8 = 256$ can be stored in a 1-byte word. The amounts of single-precision floating point ranges that can be stored in a 4-byte word depends on the number of partitions, that have to be defined in a compromise between storage space saving and information loss.

The approach of the PRESELECTIONDB provides a number of features that improve the efficiency of the data analysis with respect to the standard production of file streams:

- routines for data selection are evaluated only once during the database creation;
- the procedure to check the quality of events reduces to the verification of true/false variables;
- the same database can be used by different users to define multiple streams;
- the database can be dynamically created and integrated with new information just by re-processing the original data to check the new requirements only and pushing the result into the database;

¹This is a typical situation for the data of a HEP experiment, like the number of reconstructed tracks in a spectrometer.

²The notation includes the cases $\chi_i^{(min)} = -\infty$ and $\chi_i^{(min)} = +\infty$

- the plain database structure allow the implementation with most data analysis package;
- the use of disk storage is limited by the database size and is independent on the number of streams.

In conclusion, the PRESELECTIONDB approach allow a flexible and dynamic definition of data streams requiring less I/O resources and disk space than standard approaches.

2 Use case: PreselectionDB applied to AMS-02 data analysis

AMS-02 is a HEP experiment operating on the International Space Station, where it detect cosmic rays (mostly fully ionized light nuclei) at an average rate of 700 Hz. The AMS-02 is composed by several subdetectors, each providing different information about the crossing particle. For each cosmic ray crossing the AMS-02, the information collected by its electronic channels ($\sim 300\,000$ channels in total) is processed in flight to select the channels containing interesting information to sent to ground. The data are transferred with an average rate of 9 Mbps. At ground, for each event the electronic channel information is processed using a dedicated *reconstruction* code, that identifies patterns in the data, clusters together channels providing similar information and builds high level objects that are used by the end users to investigate the properties of cosmic rays. An example of high level object is the track reconstructed in the spectrometer or the shower reconstructed in the electromagnetic calorimeter. The number of reconstructed objects is not fixed and its content changes dynamically for each event. In the AMS reconstruction software, an additional hierarchical step is added to the data structure. After the subdetector objects are defined, they are clustered together to form a *particle* object. Such object is a collection of subdetector objects whose electronic channel information is produced by the same cosmic ray crossing the AMS-02 detector. Each event may contain 0 or few particle objects and a set of "orphan" objects, that are objects which do not belong to any particle.

In the first seven years of data taking, AMS-02 has collected more than 120×10^9 events, for a total of ~ 250 TB of raw data and close to 1 PB of reconstructed data stored on disk. Among such data set, scientists dig for interesting events that typically amount to less than 1% of the total data sample. The definition of "interesting" event strongly depends on the specific task of the analyses, and may change as the amount of collected data increases. Scientists therefore have to continuously process the whole data sample and check for each event whether a set of conditions in the form of Eq.2 is satisfied. PRESELECTIONDB

2.1 The database

For the specific application case described here, two sets of criteria have been defined:

- EVENT criteria, $g_i^{(ev)}(x_1, \dots, x_N) = 0$, based on the information common to the whole event (example: time of the event acquisition, position and pointing of the experiment, etc...);
- PARTICLE criteria, $g_i^{(pt)}(x_1, \dots, x_N) = 0$, based on the information proper of a particle object.

A *collection* is the total set of criteria for a set. The EVENT collection groups together 32 criteria and the PARTICLE collection groups together 64 criteria. The maximum number of criteria has been chosen as a compromise between a large enough variety of criteria while keeping the possibility to store the information in a 4-byte and a 8-byte word.

During the database creation all the 32 EVENT criteria, $g_i^{(ev)}(x_1, \dots, x_N) = 0$ are evaluated for each event. The results are stored using a *negative* logic in a 4-byte word (corresponding to 32 bits) where each bit maps to an EVENT criteria. The i^{th} bit is set to 1 if the criteria is *not*

fulfilled for the current event, and it is set to 0 otherwise. A similar approach is used for the `PARTICLE` criteria. For each particle object stored in the current event, the 64 $g_i^{(pt)}(x_1, \dots x_N) = 0$ criteria are evaluated and the result is similarly stored in a 8-byte word. The 8-byte words are stored in a variable size vector³.

To achieve the synchronization between the database and the data, the database also stores an unambiguous event identifier. For this particular application, the event is unambiguously tagged using a pair of numbers defined during the data-acquisition: the *run* number⁴, and a consecutive serial number (the *event* number). These two numbers are stored for each entry of the database for synchronization during the analysis procedure.

Finally, two continuous indexing variables are stored in the database: the energy measurement of the electromagnetic calorimeter *E* and the rigidity⁵ measurement of the spectrometer *R*, which are stored in the experiment data as single-point floating numbers. In the database instead, each variable range is defined using 7 bits, for a total of 128 bins. 1 bit is dedicated for the sign of the number⁶. A 2-byte word is used to store the ranges for *E* and *R* for each event. Figure 4 shows a simplified concept of the database structure for the AMS application. For this application, we have chosen to implement the database as a ROOT TTree structure,

Figure 4. Simplified concept of the database structure developed for the analysis of AMS data.

that is the same structure shared by the data. For each data ROOT file, a companion database ROOT file has been produced. Each database entry is synchronized with the data TTree entry by the *run* and *entry* branches. The `EVENT` and `PARTICLE` word and the indexing variables are stored as separate branches of the TTree

³A C++ `std::vector` for example.

⁴The *run* number corresponds to the UNIX time of the second in which the data acquisition started for the specific case of the AMS-02 experiment.

⁵The Rigidity of a particle is the ratio between its momentum *p* and its charge *q*, $R = p/q$

⁶This is useful only for the rigidity measurement *R*. The energy measurement *E* is unsigned.

2.2 The software framework

A custom software library has been written to manage the interface between the database and the data analysis. The software environment makes use of basic C++ functions only, and can be easily implemented in any other programming language. We sketch here only the basic functionalities for the database management management.

The database is created and eventually integrated with further information using a custom software interfaced with the AMS analysis libraries. The database can be stored locally or accessed remotely.

The database is accessed using the manager class `Preselection`. During the analysis, for each event the class searches the database for the entry synchronized with the current data. If the data are analyzed in increasing sequence (as typically done), this is exploited to optimize the search time.

Before the analysis, the user sets a pair of bit masks to the `Preselection` class. The masks are, in this particular case, a 4-byte and a 8-byte word that define which criteria among all those defined in the database are to be checked. The database acts as a *lookup table* for the data events.

During the data analysis, the user loops through the entries of the database and searches for interesting events. The `Preselection` class reads the `EVENT` word stored in the database entry and compares it bit by bit with the corresponding mask to check if the current events satisfies all the conditions defined by the user. Subsequently, the manager class reads the `PARTICLE` word for each particle object stored in the current event and provides the number of particles that satisfy the requirements of the corresponding mask. On the basis of this information, the user can therefore set a rule to define whether an event is "interesting" and, only at this point, read the whole event information from the file. In this context the gain with a local database and a dislocated data set on server is clear. The user has to loop locally on the database. Only if an interesting event is found based on the database information, the whole information on the event is read from the server, therefore minimizing the access to the server.

This implementation of the `PRESELECTIONDB` has been successfully used in the context of AMS data analysis to extract from data important information to study different aspects of physics publications [1–5]. This support the robustness of the approach, that could similarly implemented to the software frameworks of other experiments.

3 Conclusions

We have described an approach based on a database, `PRESELECTIONDB`, for dynamic and on-demand definition of data streams. This approach improves the flexibility and efficiency of data selection when a small subset of data has to be retrieved from a larger data set, as usually done for HEP analysis where the users are interested only in a small fraction (less than 1%) of the total data recorded by the experiment. `PRESELECTIONDB` allows users to define a wide variety of data streams without any replication of the data on the filesystem: this results in higher flexibility of the data analysis and of its upgrades, and minimizes the amount of data to be stored in local filesystems and the access to the data stored in the servers. The idea has been successfully implemented and tested in the particular case of the analysis of AMS-02 data, and it can be easily extended to any other analysis framework.

References

- [1] M. Aguilar et al. (AMS Collaboration), Phys. Rev. Lett. 113, 221102

- [2] M. Aguilar et al. (AMS Collaboration), Phys. Rev. Lett. 113, 121102
- [3] L. Accardo et al. (AMS Collaboration), Phys. Rev. Lett. 113, 121101
- [4] M. Aguilar et al. (AMS Collaboration), Phys. Rev. Lett. 117, 091103
- [5] M. Aguilar et al. (AMS Collaboration), Phys. Rev. Lett. 121, 051102