Chapter 1

The AOD format

The AOD (Analysis Object Data) format is intended to be the main file format to be used for the analysis of data produced by the experiment; both physical data and simulated Monte-Carlo data. They are the final output format for both methods of data taking, they are constructed from ESD (Event Summary Data) files containing cleaned cluster-data and reconstructed tracks and vertices. It is estimated that a yearly-median of 21 PB worth of AODs will be stored between 2020 and 2027 (excl. a 3 year shutdown between 2022 and 2025) making efficient usage and storage very important. AODs will be stored at dedicated facilities for subsequent analysis.

1.1 Analysis

Generated AODs will be stored and processed in dedicated *Analysis Facilities* or on local machines. These Analysis Facilities are specialized datacenters, part of the ALICE Computing Grid. Physicists can submit different tasks to the Grid to run against a large (or small) set of data. The Grid software automatically handles the scheduling of tasks and the IO of the AODs. Tasks may be split up over multiple machines or even multiple Grid *sites* at once, the smallest level of granularity the scheduler can handle is a single AOD file.

There are currently two ways of submitting jobs to the Grid. One can submit a single analysis task as a c++ source file, specifying the input files at run-time, or alternatively one can submit a task as part of a so-called Analysis Train. In order to do so, the analysis task has to be checked in to the central source-code repository. It is then possible to tweak the parameters and select the data of interest from a web-interface. Several tasks submitted in such a fashion (called wagons) will be gathered together by the central software reducing the required movement of data. Using analysis-trains has proven to be a much more effective utilization of resources and single-tasks are therefore being phased out in favor of analysis-trains.

Each task (or wagon as they are also called) runs as it's own separate process. Tasks can communicate between each other and other actors using a messaging system build on top of FairMQ(CITE) (which is in build on top of ZeroMQ(CITE) and nanomsg(CITE)). This messaging system can be used to share a single AOD between several tasks using shared memory or fetching data from the nearest location. for example, the data can be located starting from shared memory, then looking at a different NUMA node, then looking at other connected nodes in memory or on disk, etc., all the way to tape-storage. The exact implementation of this mechanism and topology is independent of the AOD design and Analysis Framework save for two restrictions: the AOD design has to interact (efficiently) with the fairMQ message-passing framework and the framework has to let the messaging system be in charge of memory-allocation in order to be able to use shared memory.

1.2 Requirements

Before the start of the project a list of all necessary features was compiled, these included:

- **Pruning:** The ability to reduce an AOD to a smaller subset by removal of structures and/or components of structures. This is needed in order to allow the physicists to reduce a larger dataset to a much smaller one for subsequent (local) analysis and reduce the required IO- and memory-bandwidth at runtime. For example, the most commonly used object stored in an AOD file is a detector *track*. This is the reconstructed trail left by a particle in the detector. A single track contains many different types of information such as its point-of-origin, it's momentum, the covariance matrix of the reconstruction, references to other associated objects etc. In most cases however, one is not interested in all of these components. Sometimes one might only be interested in a single component of the momentum. Pruning is the act of getting rid of all other data and, in this case, leaving only a single component of the momentum.
- **Skimming:** The ability to infer certain facts about an AOD without having to touch all the data contained therein. For example one might only be interested in tracks which have a positive charge associated with them. Skimming would be the act of reading/copying only those tracks which fit this criteria. (TODO: write psuedo-code for these to use as an example, i.e. $if(charge <= 0)\{continue;\}$)
- **Growing:** The ability to easily extend an AOD with additional data, structures or components. For example, one might like to extend the track structure with a new field to cache the result of a long and complex computation, call our hypothetical field *'bool IsHiggsBosonCandidate'*, that would be an example of growing the track structure. Another option would be the adding of a whole new struct to an existing AOD, for example one might have a set of simulated (Monte-Carlo generated) collisions and wish to extend the AOD file to include the tracks created by this simulation (note that the Monte Carlo generator produces particles which do not interact with the detector so one simulates this interaction starting from the generated particles to create a simulation of raw detector data, after which the normal reconstruction algorithms can be run to create normal AOD data).

This list was later extended to include:

- **Derived Types:** Certain components are rarely used and/or take up a large amount of disk-space when stored and can be recomputed from other stored data if needed. It is therefore favorable if these could be recomputed transparently on a as-needed basis as opposed to being stored in-file. A common example would be so called *V0s*, these are secondary-vertices[1] resulting from the decay of a neutral particle. The V0s can be recomputed from the most common data and it is therefore possible to save on storage space by not including them in the AOD files, at the cost of having to recompute them before being able to use them. By leveraging shared memory and smart disk-caching this compute penalty can be kept to a minimum.

- **Zero-copy message parsing:** Previous work(CITE,Mikolaj?) has shown that a large performance penalty is incurred by data-serialization. In order to avoid this overhead the format should be able to pass data around without any (unecessary[2]) serialization. In practice this means avoiding the use of pointers inside data-structures and keeping any-and-all data of POD-type[3].

- **Maintain a similar interface:** The current ALICE software framework is used by X(CITE) users, contains over Y lines of code, and was written over the course of Z years. While it has been deemed possible, even necessary, to convert parts of the framework to work with the new format a dramatic change in the way physicists interface with the data would involve to much man-hours in terms of rewriting existing software and utilities and forcing every user to adapt to the new framework.

- **Schema Evolution:** On occasion, changes might be made to the way in which data is stored. This might be the conversion from a single float to a 3-dimensional vector or it might be a change in coordinate system to improve accuracy and/or compression. Changes of this kind are called Schema Evolution, and it is important that the framework knows how to handle this so that it can adapt and read both old- and new-formatted data.

In addition to these given requirements the overall guidelines are such that on-disk storage space should be made as small as possible, whereas in-memory data should aim to keep the amount of bandwidth required to a minimum as the current ratio of computing power to bandwidth heavily favors compute.

[1]primary-vertices being vertices generated from a collision at the hearth of the detector and secondary vertices being those formed by the decay of particles

[2]One cannot avoid (de)serialization when passing data between machines with different native data types, such as a big-endian and little-endian machine but that is out-of-scope for the AOD.

[3]POD, or plain-old-data, is a c++ term defined in (CITE). In practice it means that a struct or class cannot contain any virtual types or define data and/or functions in more than one part of its inheritance graph (including itself).