NanoAODs

Summer student report

Lucia Anna Husová

September 6, 2017

The scientist on LHC experiment analyse a huge amount of data every day on the Grid. Thus new methods are requested, how to make the analysis more efficient. The NanoAOD is a derived dataset from AOD, where only information necessary for the analysis is stored. Thus the analysis can be more than two times faster, because of the smaller file size, which can be read faster on the Grid. The main goal of this summer student project was to help other users to start using NanoAODs by rewriting their user tasks. Two example users tasks were converted to NanoAODs and tested with the local train test. A speed up of 3.5 was reached. The results of the analysis tasks are identical independent if they use AODs or NanoAODs.

1 Introduction

At the LHC experiments, a big amount of data is taken, which should be analysed afterwards. Each analysis is specific and thus needs other information. For this reason the data taken in the ALICE experiment are stored all over the world in different formats, for example AOD. To analyse this huge amount of data lots of computing time is used[1]. To make the analysis faster a new type of dataset was developed - the NanoAODs.

The main goal of this summer student project was to help other users to start using NanoAODs by rewriting example user tasks.

2 NanoAODs

NanoAODs are duplicates of AODs on the Grid, which contain only the information about tracks and events, which is really used in the analysis. Thus the NanoAOD file size is smaller than for AODs, but these are extra files to be saved. The analysis of the NanoAODs can be multiple times faster than the one of the AODs, so CPU time is saved. The speed up depends on the fraction of the AOD information stored in NanoAODs. The disadvantage is that every user needs other information for his analysis. Thus each analysis needs special NanoAODs and that give us more files to be stored.

[1] Every day, about 600500 jobs are completed and every job takes avg. 225 CPU minutes.

The NanoAOD information is stored in three objects: AliNanoAODHeader, AliAODEvent and AliNanoAODTrack. The inherit dependance for the AliNanoAODTrack is shown in Figure 1. All the methods from AliVTrack can be used for an AliNanoAODTrack as well and there are some new functions. So if the track object is casted to an AliVTrack in the analysis task, no changes are needed and the same code can be used for AOD as well as for NanoAOD analysis.

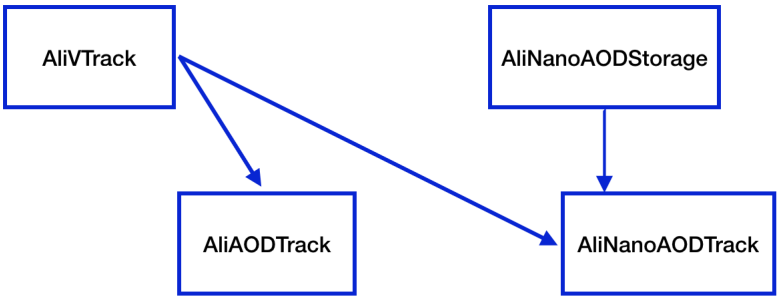

Figure 1: Inherit dependance of the class AliNanoAODTrack

2.1 Documentation

A part of this summer student project was to write a documentation of NanoADOs [1]. This Twiki Page describes how to implement new custom variables to the NanoAOD classes with examples and how to produce NanoAOD locally or on the Grid. In this page the code changes are presented, which are needed in the analysis tasks for the conversion to NanoAODs.

3 Development of NanoAODs

One of the goals of this summer student project was to develop new features for the NanoAODs and make the analysis of other users easier and faster.

3.1 Dependecies

All user classes can be dependent on the NanoAOD classes, but the NanoAOD classes must not use methods from analysis tasks.

AliNanoAODTrack functions were updated to match the equivalent AliAODTrack functions, for example the function ZAtDCA() and the track position.

3.2 New features

To make the NanoAODs to be used easier and in more analyses, new variables were implemented. Now all 21 components of covariance matrix, the filter map and the muon track identifier can be stored in the AliNanoAODTrack.

To make implementation of stored variables in the AliNanoAODHeader easier, a flexible index for each variable was implemented. It means that only the variable is added to the array, which is declared in the train configuration. Therefore, the variable indexes can change form user to user depending on the order of declaration in the train settings. It is a similar procedure as by saving variables in the AliNanoAODTrack just without using a mapping class. There are six common variables (4 types of centrality, magnetic field and run number) which can be stored. Moreover every user can define his own custom variables. For this reason an additional field was implemented in the AliNanoAODHeader, where these custom variables are stored.

The user can decide in the train configuration, if some more objects should be stored in addition to the primary vertex in AliAODEvent for the next NanoAOD analysis. A new option has been implemented to save tracks from an output array from a previous task. The name of the output array of NanoAOD tracks can be changed if it is needed. The name of both arrays should be set in the train configuration.

4 Application of the NanoAODs

4.1 Task selection criteria

If a given task is often repeated with a long CPU time then the largest CPU time saving can be achieved with the use of NanoAODs. Therefore, these were the main selection criteria for the example user tasks rewritten in this project. Two tasks were selected and thereafter converted to NanoAODs. In Figure 2, the durations of selected tasks are shown, if they were running only at one core. Both of them take enough time that the running on NanoAODs would be a valuable improvement. The statistics of the tasks is shown in the Figure 3, where is shown that both tasks run several times on the same dataset.

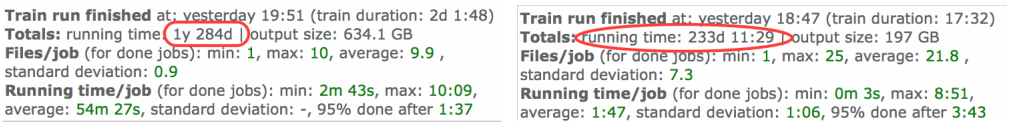

Figure 2: Average CPU of the selected tasks. Left AliAnalysisTaskCRCZDC, right AliAnalysisTaskChargedJetsHadronCF

Another selection criterium was that the task uses a limited set of variables. This ensures that the NanoAOD file size will be much smaller than the AODs, which they are generated from.

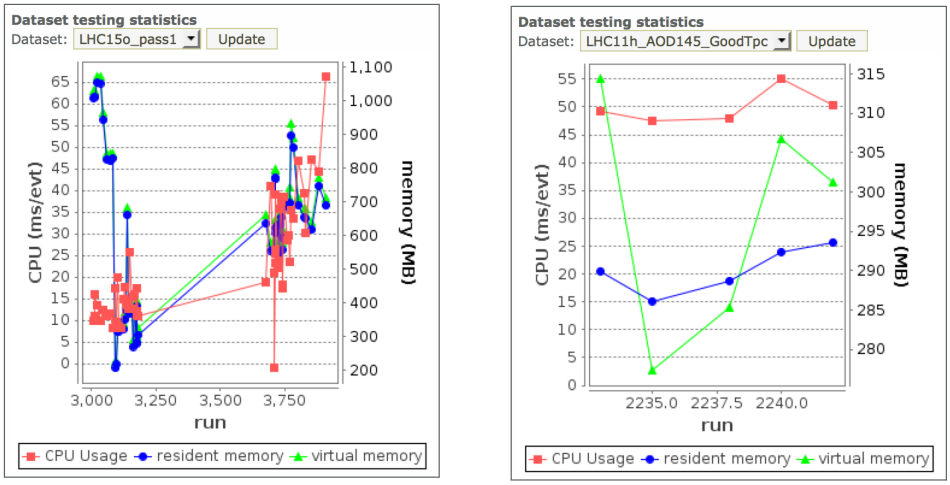

Figure 3: Using statistics of the selected tasks. Left AliAnalysisTaskCRCZDC, right AliAnalysisTaskChargedJetsHadronCF

4.2 Evaluation of Nano AODs

The analysis tasks were converted to NanoAODs and tested with the local train test[2]. The statistics for the both example tasks are shown Tables 1 and 2. It is evident, that the size of the NanoAOD files is much smaller than the size of AOD ones. The speed up increases with the number of input files. This is caused by the initial time of the task[3]. The speed up reaches 3.5 with 20 input files in the AliAnalysisTaskCRCZDC and 3.1 in the AliAnalysisTaskChargedJetsHadronCF.

From this Tables, it is obvious, that the time of NanoAOD generation is similar or even longer than the time of single AOD analysis . Thus it is really important to use the NanoAOD only in the tasks, which are repeated often. When the task run only once, there is no CPU time saving.

Number of input files	Size		Time			Speed factor
	AOD	NanoAOD	NanoAOD generation	AOD	NanoAOD	
5	4.72 GB	0.27 GB	10:25	6:39	2:49	2.4
10	8.59 GB	0.56 GB	19:25	15:57	6:04	2.6
20	18.6 GB	1.03 GB	43:35	40:08	15:40	3.1

Table 1: The statistics table of the AliAnalysisTaskChargedJetsHadronCF

[2]The tasks can be analysed on the grid with so called analysis trains. This system coordinates the ALICE user analysis in an efficient way. The local train test simulates the whole train process locally.

[3]Every task needs some time for the framework initialisation and creation of the output objects which does not depend on the analysed dataset.

Local/ Grid	Number of input files	Size		Time			Speed factor
		AOD	NanoAOD	NanoAOD generation	AOD	NanoAOD	
Local	2	0.38 GB	0.1 GB	0:30	0:31	0:15	2
Local	10	1.87 GB	0.49 GB	2:10	1:55	0:45	2.5
Local	20	4.15 GB	1.02 GB	4:25	3:24	0:58	3.5
Grid	1 run	-	-	5d 14:49	6d 9:36	3d 0:25	2.1

Table 2: The statistics table of the AliAnalysisTaskCRCZDC

Analysis with NanoAODs should give the same results as the AOD analysis. A results comparison of the analysis of the two dataset types is shown in Figures 4 and 5.

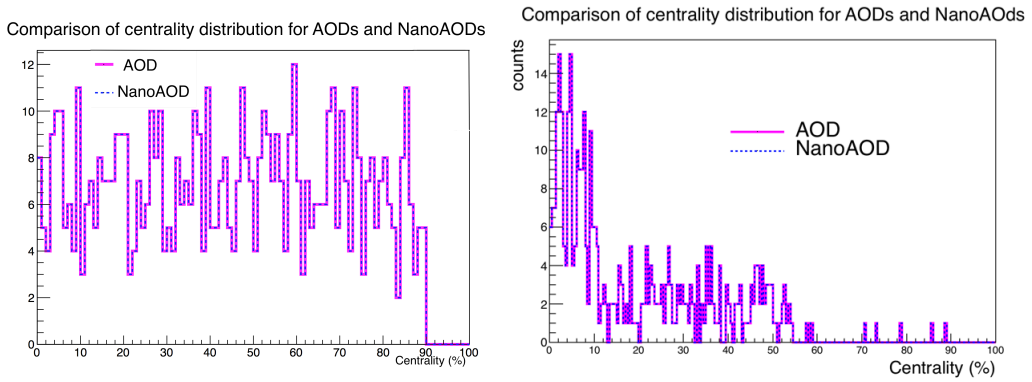

Figure 4: Comparison of centrality distribution for the example tasks

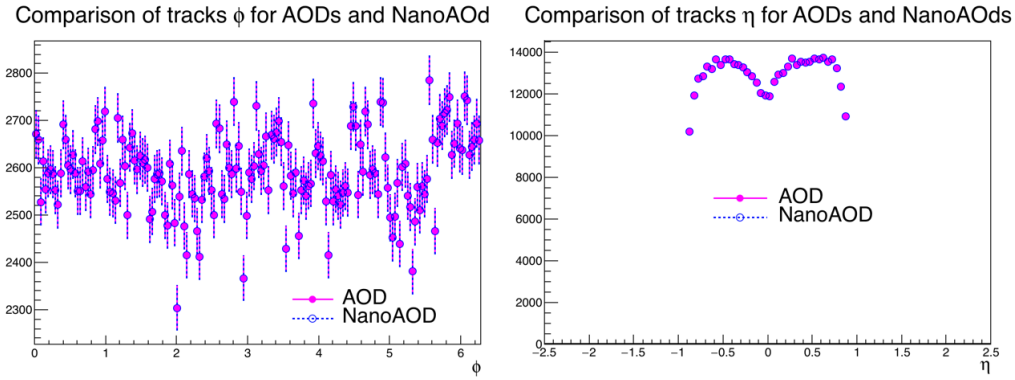

Figure 5: Comparison of ϕ and η distribution for the AliAnalysisTaskChargedJetsHadronCF

References

[1] https://twiki.cern.ch/twiki/bin/viewauth/ALICE/NanoAOD