Karlsruhe Institute of Technology

Belle II

Improvements on the basf2 validation module

Previous State – Current Developments – Future Ideas

Dr. Thomas Kuhr, Timothy Gebhard | May 12, 2014

INSTITUT FÜR EXPERIMENTELLE KERNPHYSIK (IEKP)

Previous State

Processing of the scripts

- Previous state: Scripts are processed serially, i.e. one script at a time
- Obvious problems:
  - Slow (complete validation takes 15+ hours!)
  - Waste of computing ressources $\leftarrow$ Multi-core machines are standard, but only one core is used!
- Obvious solution: **Parallelization**!
- Either on a cluster or through multi-core machines
- Main requirement: Independent tasks
- This is fulfilled in a twofold way:
  - All *packages* are independent and can be validated parallely
  - There are tasks *within a package* that are independent, i.e. several steering files generating data (like in the `tracking`-package)

Dependencies between scripts

- There are certain dependencies between the scripts, e.g. a plotting script can't be executed before a data creation script
- Previous state: Dependencies are not explicitly stated, scripts are simply executed in alphabetical order
- Problems:
  - Knowledge of the dependencies is crucial for parallel execution!
  - Adding scripts at a later point of time may be cumbersome
    - Example: You have 10 files with names `01_script`, $\cdots$, `10_script`, and you want to squeeze in a file that is executed after the 5th, but before the 6th script. $\Rightarrow$ If you want your file names to stay consistent, you will have to rename 5 of your files!
- Solution: Explicitly state the dependencies between the files!

Display of results

Validation

https://belle2.cc.kek.jp/internal/validation/index.html#

Sun May 11 15:56:36 2014

Legend:

- reference
- 10511 (current)
- 10511
- 10501
- 10483
- release-00-04-00
- build-2014-04-11

analysis

- Bd_JpsiKS,mumu_Breco
- Bd_Kstgamma_Breco
- Bu_D0pi,Kpi_Breco
- EfficiencyValidationPhotons
- EfficiencyValidationTracks
- FileEventSizes
- Hough3D
- MDSTSummary
- PIDValidationkID
- PIDValidationpiID
- RecoStats
- ResolutionValidationPhotons
- ResolutionValidationTracks
- SimStats
- Timing
- pi0Validation

arich

- ARICHValidate

cdc

analysis

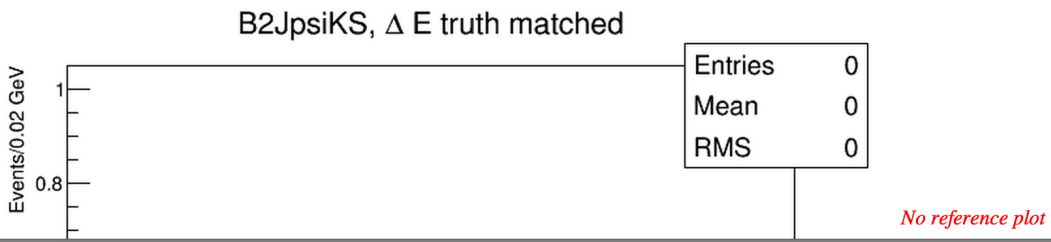
Bd_JpsiKS,mumu_Breco

No reference data

Bd_JpsiKS,mumu reco	nreco	meanerror	width	widtherror	mean
reference	n/a	n/a	n/a	n/a	n/a
current	0	84.1471	610.048	242.482	3100
10511	0	84.1471	610.048	242.482	3100
10501	0	84.1471	610.048	242.482	3100
10483	0	84.1471	610.048	242.482	3100
release-00-04-00	0	84.1471	610.048	242.482	3100
build-2014-04-11	0	84.1471	610.048	242.482	3100

Description:
Reconstruction of B -> J/psi(mu mu) K_S(pi pi) events in a signal MC sample of 1000 events, where the other side decays generically. PIDmu is not required. nreco is the total number of reconstructed, truth matched signal. The table also includes a roofit result to the J/#psi invariant mass in MeV. Tests modular analysis tools.

Check:
Ensure nreco is stable, at 400 (40 percent efficiency). The J/psi -> mu mu fit should have a Mean of 3095 MeV and Width of 5 MeV.



No reference plot

Display of results

- Previous state: Rather rudimentary display of the validation results
  - Little options to filter the results
  - No option to search for e.g. failed scripts etc.
  - Occasional problems with scaling, when plots of different basf2 versions lie within different ordners of magnitude
  - Website with results is generated directly by the `validate_basf2`-script
- Aims for the new version:
  - Improved layout (no frames?)
  - Logging functionality, which makes it easier to track down failed scripts
  - Separation of content and layout
  - Possibility to generate a PDF with all results?
  - Dynamically choose which versions are displayed?

Current Development

Overview about the current development

- Complete rewrite of the `validate_basf2`-script has begun
- Usage of the Object-Orientend Paradigm (OOP) instead of Procedural Paradigm:
  - Each script is now an object the class `Script`
  - Control of cluster/multi-processing outsourced into external classes
- Dependencies are represented by headers in the steering files
- Current state of development: The new version of the script can
  - read in a directory
  - collect all steering files in there
  - read out the dependencies from the file header
  - execute the scripts parallelly, either on a cluster or locally by spawning new processes for each script
- Performance gain for validation of the `tracking`-module: About **70%**

Header Information

- As mentioned above, it is useful to explicitly state the dependencies between the steering files
- There are two major ways of realizing that:
  - One file that holds all dependency information for a module (similar to a makefile)
  - Provide each steering file with a header which contains the dependencies of that script
- We chose the latter option, because this allows to easily store a variety of meta-information about the script, such as
  - the dependencies of the steering file
  - the files generated by the steering file (useful to check if script was executed properly)
  - the author/a contact person for the script
  - . . .

The file headers

- Meta-information are given by keywords in the file header, which is parsed by the `validate_basf2`-script:

```
# @StartHeader (this is technically just decoration)
#
# $AUTHOR = John Doe
# $DATE   = 2014-05-12
# $DEPENDENCIES = some_steering_file.py, another_file.py
# $OUTPUT = some_data_file.root
#
# @EndHeader
```

- As of now, there are four keywords: `$AUTHOR`, `$DATE`, `$DEPENDENCIES`, `$OUTPUT` $\rightarrow$ **What else would be useful?**
- **How to deal with modules that do not (yet) have file headers?**

About the parallelization

- Multi-processing parallelization is realized via Python's `subprocess`-module
- Allows to spawn a new process for each script, avoiding the Python Global Interpreter Lock (GIL)
- Parallel execution on a cluster dependes heavily on the specific cluster infrastructure $\Rightarrow$ External class which provides the controls, and needs to be written anew for every new cluster
- Default option will be multi-processing parallelization
- As for the parallelization algorithm, several approaches have been considered

The parallelization algorithm

- Currently, we use some kind of “heartbeat algorithm”:
  - Every second, there is a “heartbeat”-signal
  - This signal calls a functions, which loops over all `Script`-objects, i.e. all steering files
  - It checks if all other scripts on which a script depends have been executed already
  - If so, the script is executed
  - If a script is flagged as *running* already, it checks whether execution has finished
  - If so, the corresponding `settled_dependencies`-lists are updated
- This approach was chosen because it
  - is reasonably fast (little idle-times)
  - is thread-safe (only one process accessing a `Script`-object at a time!)
  - gives reasonably well-defined states of the program (debugging!)

Future Ideas

Future Ideas/To-Do-List

- Dividing the steering files in three levels:
  - **Plotting** $\rightarrow$ Fast, no parallelization necessary
  - **Production** $\rightarrow$ Generation of data files, parallelized
  - **Release** $\rightarrow$ Generation of high statistics data files, very big data amounts, only executed once a month
- Display of failed scripts
- Contact adress for plots
- Expert plots (expert directory in root file)
- Selection of plots and versions on the web interface
- ROOT JavaScript Interface on the web interface?

More ideas are very welcome!