AliROOT benchmarking and QA

Mikołaj Krzewicki, FIAS

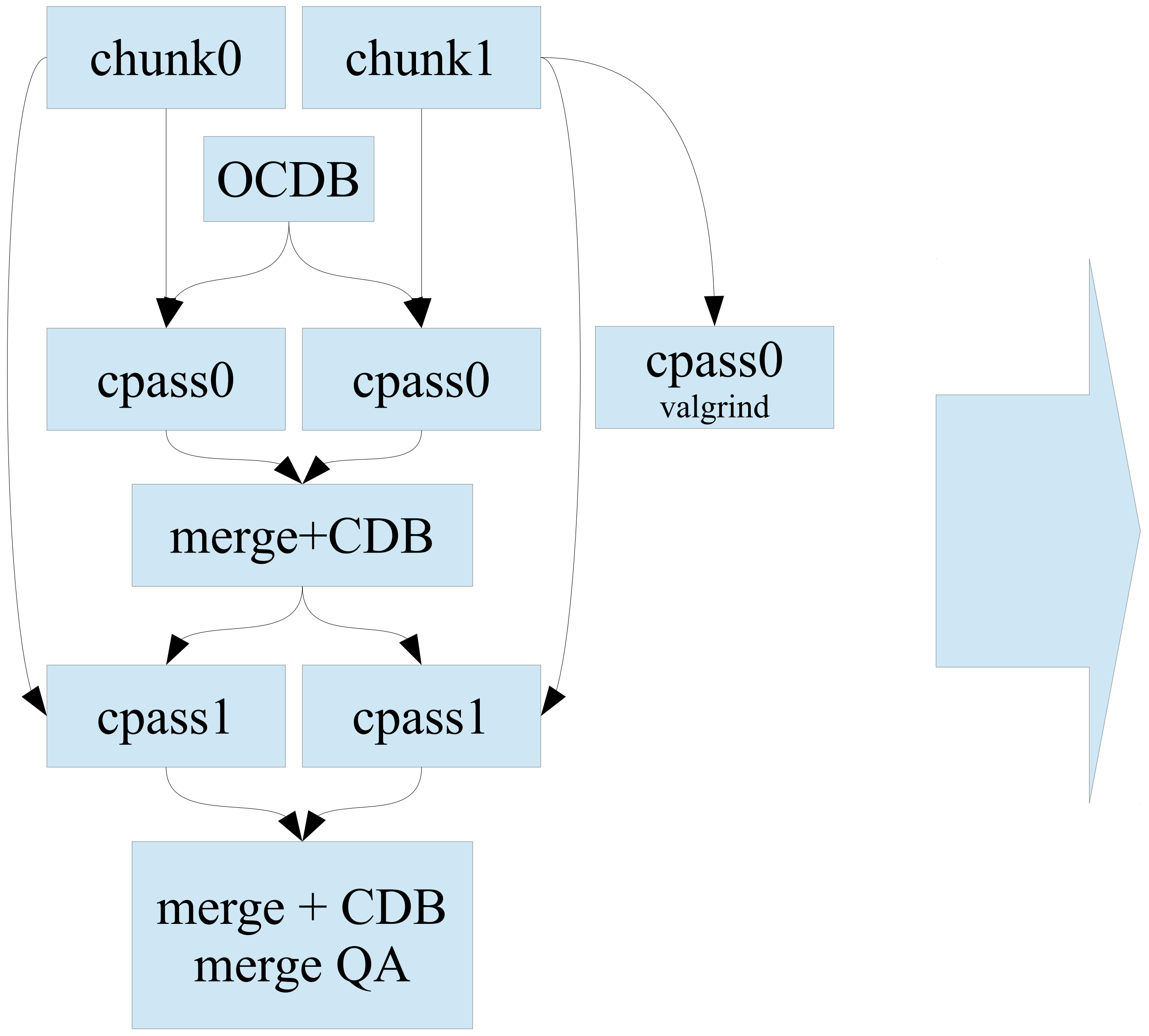
testing the entire chain

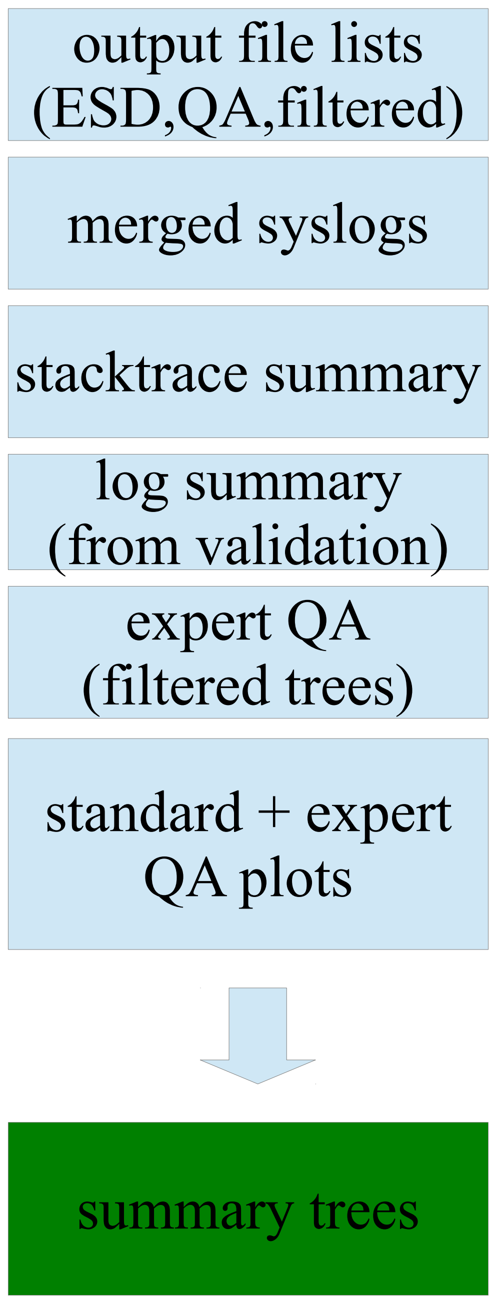
- all of the following is based on expert experience with developing/debugging/running/..., discussions with other detectors, offline, etc...
- people involved/contributing: Jan Wagner, Marian, MK. (core), plus many more provide valuable contributions: Michael Knichel, Jakob Blomer, Predrag, Dario,...
- need to stress the entire chain to test all components
- full cpass0/cpass1/QA scheme on reference data
  - reconstruction
  - calibration
  - analysis and QA frameworks
  - **software performance**: syswatch, valgrind, logs, ...
  - **physics performance**: standard+expert QA
- Keep it independent of the grid to keep everything consistent
- test an internally consistent release:
  - benchmark + all helper scripts part of release

benchmark procedure

- Input data: raw filtered data (enhanced high pt, V0, cosmics
  - LHC10e + LHC11h (~100 runs,2k files) – smaller selection to be made by detector experts for running on a smaller system (~200 fi
  - On the grid: alien:///alice/reference
  - Copied to EOS public for direct access from the nodes
  - If directory structure is “standard”, i.e. (.../2010/LHC10e/000137234/...) full automation (run number, year extracted from path), otherwise each run has to be handled manual
- OCDB (cvmfs)
- Software (cvmfs)
- Storage space for the output (/eos)

benchmark, schematic




output file lists (ESD,QA,filtered)
merged syslogs
stacktrace summary
log summary (from validation)
expert QA (filtered trees)
standard + expert QA plots
summary trees

Output: QA

- Standard QA plots by central script as used for the QA webs
(e.g. for TPC http://aliqatpc.web.cern.ch/)
- Output the same as for production – a directory structure wit
plots and html files
  - easy to publish, check and compare
- Expert QA: filtered trees
  - Plot generation also handled by the standard QA script

Output: summary trees (expert)

- combine various outputs/status into one tree
- extremely useful for understanding performance/problems
- in the tree (right now):
  - Cpass0 status variables
  - Cpass1 status variables
  - TPC QA trending variables
  - TPC calibration Cpass0 variables
  - TPC calibration CPass1 variables
- Visualization can be easily implemented.

backup

How it works

- Bash script: `$ALICE_ROOT/PWGPP/benchmark/benchmark.sh + config example: $ALICE_ROOT/PWGPP/benchmark/benchmark.config`
- Works on batch systems (LSF,SGE) – as used at GSI
- Also: using makeflow for job distribution and dependencies (`http://ccl.cse.nd.edu/software/makeflow/`)
  - Make-like configuration (target: sources)
  - some limitations (from out point of view):
  - hard flow dependencies – if one job fails entire flow stops
  - we catch it: from makeflow perspective all finishes
  - job wrappers (almost) always exit normally
  - decision whether or not to run the actual cpu intensive task based on input (cpass1 wrappers always run, actual cpass1 only runs when cpass0 produced calib, etc...)

how it works (contd)

- configuration contained in a config file
- many options, flexible
- config file options can be overridden on the command line
  - e.g. fixed config file for routine testing, override the aliroot env script on the command line.
- easily override most steering scripts by default taken from your aliroot (e.g. merge.C, rec.C, etc.)
- new feature: MC using the benchmark script.
  - recently implemented by Jan, reported to be working, might still need some development.

simple HOWTO

```
vi benchmark.config
```

edit config

```
./benchmark.sh run test1 benchmark.list benchmark.config ocdbStorage=”raw://”
```

arbitrary production ID

config file

command

List of input files

override config options

QA tool requirements

- bash version >= 4.0
- all logic: scripts, macros taken from $ALICE_ROOT + standard linux utils
- input files follow path convention: prefix/dataType/year/LHCperiod/pass/ (e.g. /prefix/sim/2013/LHC13b/pass1/)
- one input file per run (as in the standard QA schema)
- **to participate: provide a script for plot creation/trending!**
- detector scripts have to be named according to a convention as the detector name is derived from the script name
- for central running a fixed output directory structure was created, constraints on the naming for scripts:
- {ITS,TPC,TRD,TOF,HMP,PHO,EMC,V0, T0,FMD,PMD,MU,ZDC,PID, TRK,EVS,CAL}.sh

QA tool, input&output

- the tool: `$ALICE_ROOT/PWGPP/QA/scripts/runQA.sh`
- detector scripts: `$ALICE_ROOT/PWGPP/QA/detectorScripts/DET.sh` `(TPC.sh, TOF.sh, TRK.sh, T0.sh, etc...)`
- default input: file list of standard Qaresults.root (in fact by default a path to /.../root_archive.zip#QAresults.root)
- output: directory structure per detector with output per run/production
  - per run the detectors produce plots + a trending file (trending.root with a tree called “trending” inside) + custom information(logs)
  - default trending.root is provided if detector does not produce it (`$ALICE_ROOT/PWGPP/macros/simpleTrending.C,` basic stats for all histograms in Qaresults.root for given detector)
  - per production (period/pass) detector produces trending plots / period wide QA (merged trending.root is provided by system)

QA tool, input&output

- safe updates (final location only updated after log validation
- log validation:stdout of detector scripts + any *.log files validated – in case of trouble automatic notification of QA responsible + temporary output still available
- automatic summary.log (per detector)
- period level QA is rerun if a run is updated

QA tool requirements for detector scripts

- detector scripts define 2 functions (see template `$ALICE_ROOT/PWGPP/QA/detectorScripts/`EXAMPLE.sh)
  - runLevelQA, input here is the qa file (Qaresults.root), path provide externally, together with some other variables, like the run number
  - periodLevelQA – merged trending.root file provided by framework and present in the running dir
- run in the current directory, it will be created for you, leave a output there

detector QA script example

- simple edit of the example: PWGPP/QA/detectorQAscripts/EXAMPLE.sh.template
- T0.sh:

```
#available variables:
#  $dataType     e.g. data or sim
#  $year         e.g. 2011
#  $period       e.g. LHC13g
#  $runNumber     e.g. 169123
#  $pass         e.g. cpass1,pass1,passMC

runLevelQA()
{
  qaFile=$1

  cp $ALICE_ROOT/T0/MakeTrendT0.C .
  aliroot -b -q -l "MakeTrendT0.C(\"$qaFile\",${runNumber})"
}

periodLevelQA()
{
  trendingFile=$1

  cp $ALICE_ROOT/T0/drawPerformanceT0QATrends.C .
  aliroot -b -q -l "drawPerformanceT0QATrends.C(\"$trendingFile\")"
}
```

QAtool, local/expert use

- in principle all configurable options go to a config file, but defaults are sane and options can be provided (or overridden) via command line. (run the scripts wihtout args to see some basic docs)
- `./runQA.sh inputList=qa.list`

- `./runQA.sh inputList=TPCfiles.list includeDetectors=TPC`
- `./runQA.sh inputList=TOFfiles.list includeDetectors=TOF`
- `./runQA.sh configFile=/path/to/config.file`
- output is then in the current directory, one dir per detector