Online software development

Antonín Květoň

COMPASS Front-End, Trigger and DAQ Workshop
3 March 2020, CERN

Contents

News from CERN IT

- IT-SME NA Coordination meeting: hardware refresh campaign (500+ servers being retired)
- This affects our VMs (COMPASSVM03 and COMPASSVM20) – because of network constraints, CERN IT won't migrate them for us
- Migration be done during 2020 – the hardware will be decomissioned at the end of the year
- c.f. `https://indico.cern.ch/event/864523/`

What’s running on our VMs?

- COMPASS Web
- Run logbook
- Elog
- MySQL replication (to $\sim$5 institutes)
- Production DB

Plan for 2020

- All of the above items will stop working if we do not go through with the migration during 2020!
- Jan Tomsa to be present for 2 weeks sometime during March-May
- The plan: leverage the situation, potentially also migrate our VMs to containers (Docker/Kubernetes)
- This would have the added benefit of removing blockers to logbook migration – CERN webservices cannot run root, which we need for coool histograms (although CERN IT still won't like that, they want us using webservicers)
- Note: the IPs may change – a mail will be sent out when this happens

CALS API changeover

- CERN Accelerator Logging Service (CALS) – Extraction API to be switched from CALS to NXCALS
- CALS API: Java
- NXCALS API: Java backport for CALS, Spark (Python, Scala, Java)
- The plan is for CALS to be phased out by the end of 2020
- c.f. `http://nxcals-docs.web.cern.ch/0.4.11/` and `https://wikis.cern.ch/display/NXCALS/CALS+to+NXCALS+Backport+API+Status`

Where do we use the CALS API?

- Start of run scripts – more on this later
- Any other users?

Recap of projects done in 2019

- XSwitch integration polish
- Improved configurability
- UDP monitoring
- SLC6 $\rightarrow$ CC7 upgrade
- New utilities for compilation and deployment
- New server for remote connectivity
- New CR hardware (6 new PCs)
- Zabbix integration revamp (WIP)
- Common Client Core (WIP)
- Start of run Scripts phase 1

Ongoing Student projects

Aleš Suchomel

- Taking over DIALOG development
- The main goals include simplifying the API, streamlining process-to-process messaging and improving the system robustness

Ongoing Student projects

Lucie Roškotová

- Reworking the DAQ configuration interface in accordance with modern HCI principles
- Will perform low-fidelity prototype usability testing during this dry run

Ongoing Student projects

František Voldřich

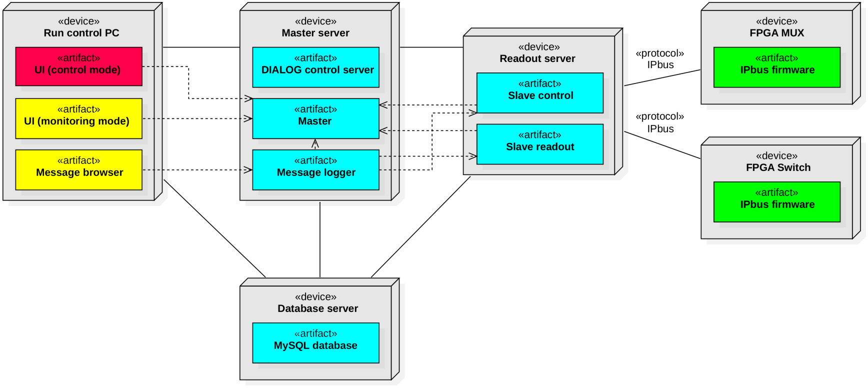
- Public COMPASS web
- Improved storage of publications – loading from CDS, administration of posts, pictures...

Ongoing Student projects

Tomáš Brabec

- WebSockets DIALOG daemon
- The main goal is to allow web applications to connect to DIALOG services

DAQ deployment diagram



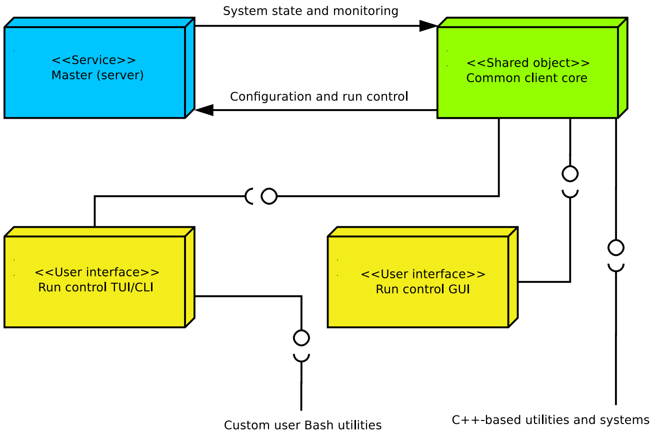
Master-GUI communication

- The Master publishes aggregated information regarding the DAQ and its processes on DIALOG services, most notably the STATUS_OF_SLAVES service
- Several processes connect to these services (GUI, CLI, EventSizeDisplay)
- However, there is no unified library that would connect to these service, parse the messages and represent the information in an object-oriented model – each of the processes used its own implementation

Common client core

- A new common library was created in 2019, still a work in progress
- New DAQ structure library created as a sub-project (already being used in the DAQ rawfile generator)
- Separate model and view logic (MVC design pattern) – the library becomes an API to the DAQ for free!

New interface architecture



API example

```
#include <QtCore>
#include "daqdebugger.h"
#include "../compass-rccars-daq-interface-common/backend.h"

int main(int argc, char *argv[])
{

    //QT setup
    QCoreApplication* app = new QCoreApplication(argc, argv);

    //DAQ configuration loading
    config_consts::get_instance().read_consts(app->arguments().at(1));

    //DIM setup
    int MasterPort = config_consts::get_instance().master_port();
    DimClient::setDnsNode(config_consts::get_instance().master_address().toStdString().c_str(), MasterPort);

    //DAQ debugger setup
    DAQDebugger::init(argv[0], "uiTEST");

    //Common client core setup
    backend::i().init_dialog(app);
    backend::i().start_communication();
    backend::i().detach_model();
    backend::i().connect_communication_to_model();

    //Access the multiplexer model
    while(1){
        QTextStream(stdout) << backend::i()._model.get_slinks().MUXES().at(0).current_spill_number();
        sleep(1);
    }

    return app->exec();
}
```

Current status

- A large part of the monitoring functionality is implemented, but not yet properly tested
- Control functionality is still yet to be implemented
- Unfortunately, the project is not the highest priority, but we would like to finish the common client core and integrate it into the DAQ GUI before the dry run

Start of run scripts rework – contents

- Current status
- Potential achitecture changes before dry run 2020
- Discussion with logbook users and detector experts on further development

What are the "Start of run scripts"?

- We use the term "Start of run scripts", or "SOR" to refer to functionality provided by a collection of legacy user programs, which the iFDAQ software calls at the beginning AND the end of a run in the form of plain C system() calls or QProcess() calls
- The purpose of these scripts is almost exclusively logkeeping, i.e., inserting metadata about runs into the logbook database, which are then available through the Run Logbook
- Some of these programs may be as old as COMPASS itself – over the years, they have evolved into quite the zoo of languages (Bash, Tcl, Perl, Java, Python)

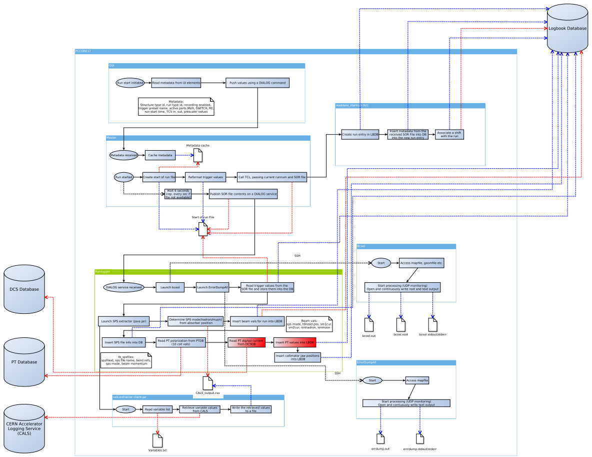
Start of run scripts rework motivation

- Little to no interoperability between C++ and the listed languages
- Our software called these scripts asynchronously and then forgot about them
- Even though this could cause a plethora of problems in theory, it actually worked quite well in practice
- However, such a codebase would be extremely difficult to maintain going forward → we needed to unify the programming language! (and include the SOR functionality in our version control)
- Thus, it was decided the functionality would have to be rewritten in C++

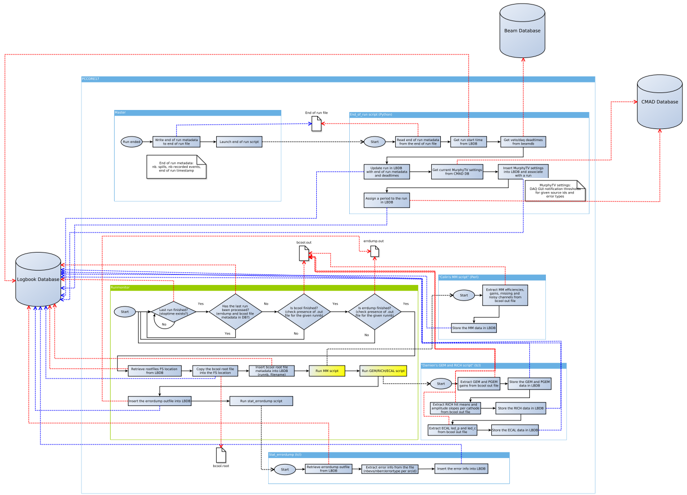
Start of run scripts rework status

- 2018, 2019 – about 6 man-months put in by L. Roškotová, A. Květoň, M. Bodlák
- A major part of the old codebase migrated to C++ and succesfully tested during the 2019 dry run
- However, we are not quite there yet...
- Some old code still remains (do we need it? → discussion)
- Language migration is one thing, but what about architecture?

Start of run scripts – current status


Logbook Database
PCCORE17
GUI
Run start initiated
Read metadata from UI elements
Push values using a DIALOG command
Metadata:
Structure type id, run type id, recording enabled,
trigger preset name, active ports MUX, SWITCH, RE,
run start time, TCS in, out, prescaler values
Master
Metadata cache
Metadata received
Cache metadata
Run started
Create start of run file
Reformat trigger values
Call TCL, passing current runnum and SOR file
Wait 6 seconds
(rep. every sec if
file not available)
Publish SOR file contents on a DIALOG service
Start of run File
readdate_startrun (tcl)
Create run entry in LBDB
Insert metadata from the
received SOR File into DB
into the new run entry
Associate a shift
with the run
SSH
Runlogger
DIALOG service received
Launch bcool
Launch ErrorDumpAll
Read trigger values from the
SOR file and store them into the DB
Launch SPS extractor (java jar)
Determine SPS mode(hadron/muon)
from absorber position
Insert beam vals for run into LBDB
Beam vals:
sps mode, t6head pos, sm1cur,
sm2cur, ionhadron, ionmuon
Insert SPS file info into DB
Read PT polarization from PTDB
(10 coil vals)
Read PT dip/sol current
from DCSDB
Insert PT values into LBDB
Insert collimator jaw positions
into LBDB
tb_spsfiles:
spsfileid, sps file name, bend refs,
sps mode, beam momentum
DCS Database
PT Database
CERN Accelerator
Logging Service
(CALS)
CALS_output.csv
cals-extractor-client.jar
Start
Read variable list
Retrieve variable values
from CALS
Write the retrieved values
to a file
Variables.txt
Bcool
Start
Access mapfile, geomfile etc.
Start processing (UDP monitoring)
Open and continuously write root and text output
bcool.out
bcool.root
bcool.stdout/stderr
SSH
ErrorDumpAll
Start
Access mapfile
Start processing (UDP monitoring)
Open and contiuously write text output
errdump.out
errrdump.stdout/stderr

End of run scripts – current status


Beam Database
CMAD Database
PCCORE17
Master
Run ended
Write end of run metadata to end of run file
Launch end of run script
End of run metadata: nb. spills, nb recorded events, end of run timestamp
End of run file
End_of_run script (Python)
Start
Read end of run metadata from the end of run file
Get run start time from LBDB
Get veto/daq deadtimes from beamdb
Update run in LBDB with end of run metadata and deadtimes
Get current MurphyTV settings from CMAD DB
Insert MurphyTV settings into LBDB and associate with a run
MurphyTV settings: DAQ GUI notification thresholds for given source ids and error types
Assign a period to the run in LBDB
Logbook Database
bcool.out
errdump.out
Runmonitor
Start
Last run finished? (stoptime exists?)
No
Yes
Has the last run been processed? (errdump and bcool file metadata in DB?)
Yes
No
Is bcool finished? (check presence of .out file for the given runnb)
No
Yes
Is errdump finished? (check presence of .out file for the given runnb)
No
Yes
Retrieve rootfiles FS location from LBDB
Copy the bcool root file into the FS location
Insert bcool root file metadata into LBDB (runnb, filename)
Run MM script
Run GEM/RICH/ECAL script
Insert the errordump outfile into LBDB
Run stat_errordump script
bcool.root
"Colin's MM script" (Perl)
Start
Extract MM efficiencies, gains, missing and noisy channels from bcool out file
Store the MM data in LBDB
"Damien's GEM and RICH script" (tcl)
Start
Extract GEM and PGEM gains from bcool out file
Store the GEM and PGEM data in LBDB
Extract RICH hit means and amplitude slopes per cathode from bcool out file
Store the RICH data in LBDB
Extract ECAL led_p and led_c from bcool out file
Store the ECAL data in LBDB
Stat_errordump (tcl)
Start
Retrieve errordump outfile from LBDB
Extract error info from the file (nbevs/nberr/errortype per srcid)
Insert the error info into LBDB

What does the future hold for the architecture?

1. The new scheme will completely eliminate the GUI from the picture
   - Sending information back and forth is nonsensical and convoluted
   - For this, the master will have to connect to the TCS DIM services and include some additional parsing logic
   - As an added bonus, the GUI (common backend) will no longer be dependent on DIM and communicate solely through DIALOG

What does the future hold for the architecture?

2. The Start/End of run files will be removed from the system
   - At this point, the only reason for the Start/End of run files is the legacy architecture
   - This has also caused synchronization problems during the dry run
   - The metadata will be sent over DIALOG in a new format instead
3. Readdate_startrun will be rewritten into C++
   - This is simply just a piece of code that has not been rewritten yet
   - The functionality will be integrated into the master process

What does the future hold for the architecture?

4. We will migrate from CALS to NXCALS
   - This is a little bit complicated, as no C++ API exists (and there are no plans for it in the future)
   - Potential solution: write the extractor in Python and call the functions in C++ using the Python C API → interoperability for free!
   - This would also remove the filesystem dependency, as the functionality would be integrated into the Runlogger process
   - Alternative: use the Java backport API and keep going as we have until now

What does the future hold for the architecture?

5. We will remove the DCS DB dependency
   - Accessing Oracle DB using Qt on Linux is a massive headache
   - We only read two values (PT dipole/solenoid current)
   - We will read those from an alternative source (to be discussed)
   - An alternative is to write a Python accessor and use the Python C API again (or use a non-Qt C++ solution)
6. We will rewrite the stat_errordump script into C++
   - Just another piece of code that has not been rewritten yet
   - The functionality will be integrated into the runmonitor process

What does the future hold for the architecture?

7. We will servicize Runlogger and Runmonitor
   - The processes are currently being kept alive by the Master process
   - Better to have them run as self-recovering services
   - This is in line with a general transition from processes to services – this will also affect the Master process, the DIALOG server and the Message Logger
8. ErrorDumpAll, bcoool API?
   - A big questionmark, since this would involve a non-trivial amount of work
   - This would allow us to integrate their functionality directly into runlogger and remove the filesystem dependency
   - Maybe in 2022?

Open questions – do we really NEED this functionality?

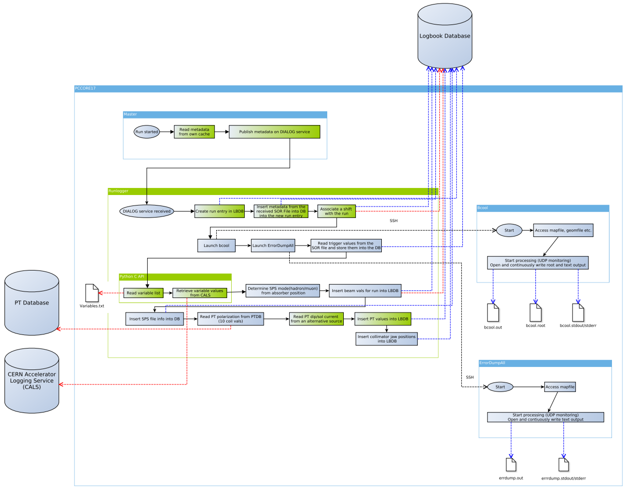
- A lot of the code is very old – not having to reimplement it would save manpower

1. "Damien's script" – RICH part
   - runlb.tb_RICH1data – last entry containing hit means and amplitude slopes per cathode from run 45001, which is 2006
   - Can we get rid of it?
2. "Colin's MM script"
   - runlb.tb_MMdata – last entry containing efficiencies, gains and missing and noisy channels from 2014
   - Can we get rid of it?

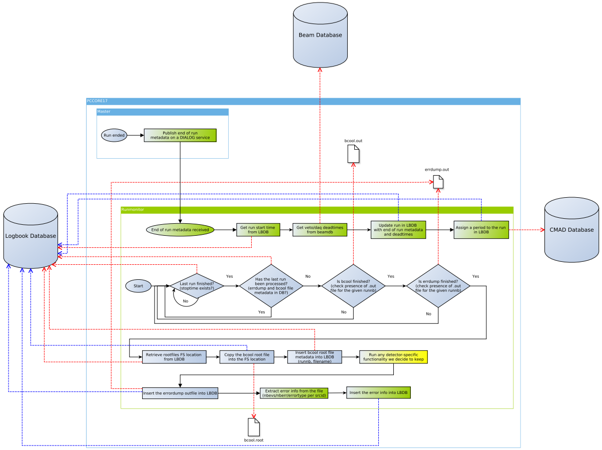
Open questions – do we really NEED this functionality?

3. "Damien's script" – GEM, PGEM and ECAL parts
    - Last entries in tb_gemdata and tb_ecaldata from the dry run, so this code seems to be working just fine
    - Is the data still being used? If so, we would ask the GEM and ECAL experts for assistance with rewriting this part of the code into C++.
4. MurphyTV settings
    - We save and display MurphyTV notification thresholds per run in the logbook
    - This is the notification that pops up in the main DAQ GUI
    - Is there a good reason to store this history per run? Can we get rid of it?

Start of run scripts – proposed architecture


Logbook Database
PCCORE17
Master
Run started
Read metadata from own cache
Publish metadata on DIALOG service
Runlogger
DIALOG service received
Create run entry in LBDB
Insert metadata from the received SOR File into DB into the new run entry
Associate a shift with the run
SSH
Launch bcool
Launch ErrorDumpAll
Read trigger values from the SOR file and store them into the DB
Python C API
Read variable list
Retrieve variable values from CALS
Determine SPS mode(hadron/muon) from absorber position
Insert beam vals for run into LBDB
Variables.txt
Insert SPS file info into DB
Read PT polarization from PTDB (10 coil vals)
Read PT dip/sol current from an alternative source
Insert PT values into LBDB
Insert collimator jaw positions into LBDB
PT Database
CERN Accelerator Logging Service (CALS)
Bcool
Start
Access mapfile, geomfile etc.
Start processing (UDP monitoring) Open and continuously write root and text output
bcool.out
bcool.root
bcool.stdout/stderr
ErrorDumpAll
SSH
Start
Access mapfile
Start processing (UDP monitoring) Open and contiuously write text output
errdump.out
errrdump.stdout/stderr

End of run scripts – proposed architecture


Beam Database
PCCORE17
Master
Run ended
Publish end of run metadata on a DIALOG service
bcool.out
errdump.out
Logbook Database
Runmonitor
End of run metadata received
Get run start time from LBDB
Get veto/daq deadtimes from beamdb
Update run in LBDB with end of run metadata and deadtimes
Assign a period to the run in LBDB
CMAD Database
Start
Last run finished? (stoptime exists?)
No
Yes
Has the last run been processed? (errdump and bcool file metadata in DB?)
Yes
No
Is bcool finished? (check presence of .out file for the given runnb)
No
Yes
Is errdump finished? (check presence of .out file for the given runnb)
No
Yes
Retrieve rootfiles FS location from LBDB
Copy the bcool root file into the FS location
Insert bcool root file metadata into LBDB (runnb, filename)
Run any detector-specific functionality we decide to keep
Insert the errordump outfile into LBDB
Extract error info from the file (nbevs/nberr/errortype per srcid)
Insert the error info into LBDB
bcool.root

Free-running DAQ format: what needs to be changed in online software?

- The concept of "event" will be abandoned in the software leading up to the trigger nodes
- The majority of our online packages are strongly based on this concept
- The following slides are an educated guesstimate of how severely the different packages will be affected by the changes, leaving out the obvious major necessary changes to the Slave Readout process

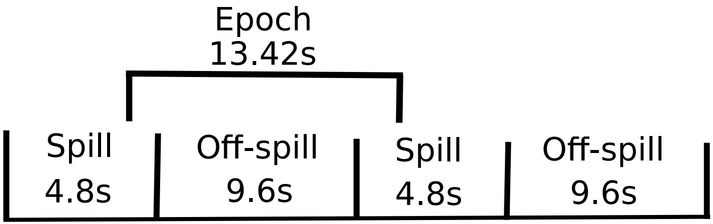
Free-running DAQ: what needs to be changed in online software?

1. A large portion of the readout/synchronization logic depends on S-Link "event types" (first in run, first in spill...)
   - **Packages affected**: Master, Slave Readout
   - **Impact**: low if we keep this information in the S-Link global header, very high if we do not
2. Some TCS DIM services will be altered
   - This information is also to be supplemented by HLT monitoring information
   - **Packages affected**: Master, GUI/CLI/Common Backend, Runlogger, Runmonitor
   - **Databases affected**: Logbook Database
   - **Online tools affected**: Logbook
   - **Impact**: Low to medium

Free-running DAQ: what needs to be changed in online software?

3. What is going to happen to the concept of a spill in terms of recording and bookkeping?
   - Currently, we take runs of 200 spills each. In the new format, this distinction might become less obvious – an epoch may end in the middle of an spill, which would require us to define how we handle this at the end of a run.
   - We could also make runs epoch-based as opposed to spill-based
   - **Packages affected**: Master, Run Logger, Run Monitor, Slave Control, Slave readout
   - **Databases affected**: Logbook Database, RCCARS database, RCCARS_log database
   - **Online tools affected**: Logbook
   - **Impact**: Medium to high

End of run bookkeeping problem

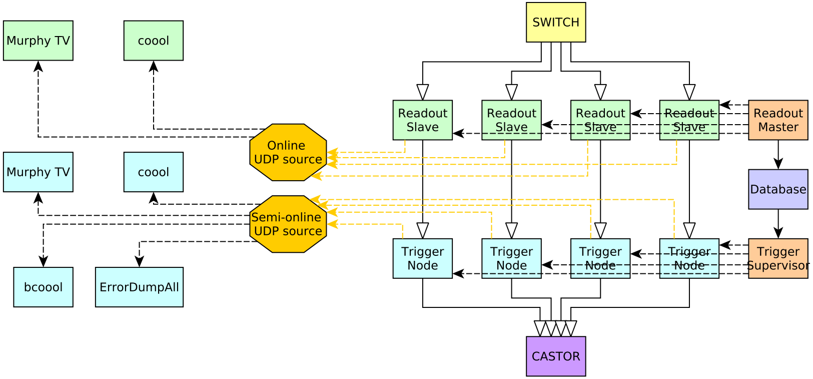


4. Almost all of our packages operate with the concept of an event number/event size $\rightarrow$ some semantical refactoring is required to slice number/slice size
   - **Packages affected**: Master, GUI/CLI/Common Backend, EventSizeDisplay, Slave Control, Slave Readout, Message Logger, Message Browser, RunMonitor, RunLogger
   - **Databases affected**: Logbook Database, RCCARS_log database
   - **Online tools affected**: Logbook
   - **Impact**: low to medium, based on what we decide to do with the databases

Monitoring in free-running DAQ

- Monitoring is a different beast entirely
- For HLT-only mode, we may need two concurrent variants: Pre-HLT monitoring (online) and Post-HLT monitoring (semi-online)
- Pre-HLT monitoring: we pick slices from the Slave Readout process at random with user-defined probability
- Post-HLT monitoring: we pick a portion of "event candidates" from the HL trigger nodes for monitoring
- If we want to keep the monitoring scheme as it is now, we need to impose additional constraints on the HLT nodes and streamline the dataflow, otherwise it won't be "online monitoring" in the true sense

Monitoring in free-running DAQ



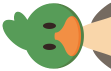
Monitoring in free-running DAQ

In short, the current proposed variants are:

1. Random pre-HLT selection
2. Post-HLT monitoring (with tighter trigger node requirements)
3. Random pre-HLT selection with post-HLT monitoring
4. Prescaled online HLT nodes in readout layer (with strict deployment requirements)

Monitoring in free-running DAQ

- Either way, the impact is very high
- DaqDataDecoding library needs to be rewritten
- MurphyTV, coool, bcoool and ErrorDumpAll need to be adapted (the exact extent is still to be investigated)



Thank you!