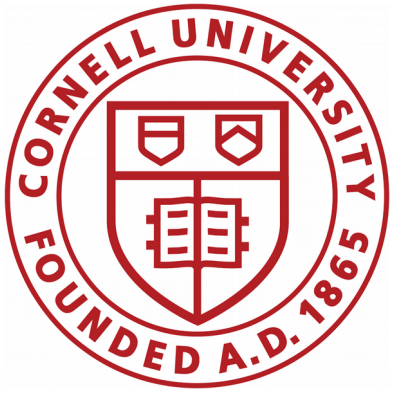


CBPM3 10 Hz continuous monitoring: database and data/metadata storage

Antoine

CBPM3 meeting

Sep 14, 2022

CBPM3 10 Hz continuous monitoring

Continuous monitoring of the beam position:

- ✗ data acquisition triggered at 10 Hz
- ✗ data corresponds to many turns averaged together (configurable)
- ✗ up to ~30,000 turns averaged per trigger
- ✗ data/metadata stored in database for online/offline monitoring/analysis
- ✗ want easy analysis of past data with proper metadata information (pedestal, gain, quad offset etc.) → need information history

Question: does the continuous monitoring need the ability to average several or all trains/bunches (Libera-like), or are we just interested in a specific train and bunch?

Data, metadata and database

CBPM3 has a dedicated PostgreSQL database to store data and metadata

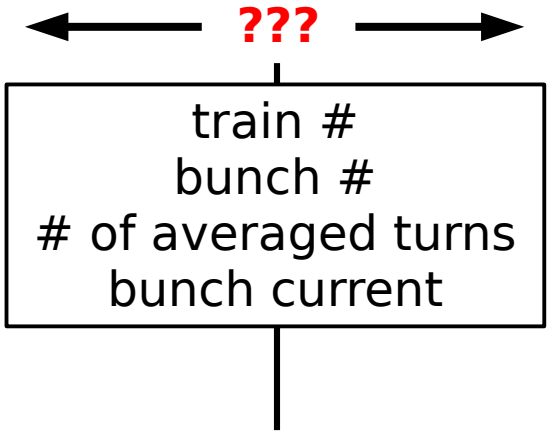
data (fast changing): information collected/stored for every acquisition trigger

metadata (slow changing): information/configuration collected/stored/monitored on a per-need basis

Since PostgreSQL is a structured database, we need to freeze a data/metadata scheme and think as far ahead as possible:

- ✗ start collecting history from day 1
- ✗ optimize data types and estimate storage needs for the years to come

Data	Metadata
button mean	button pedestal
button RMS	button gain
trigger timestamp	quad offset
module location (e.g. 22E)	gain settings?
	timing settings?
	board temperature?
	board hardware revision?
	firmware version?
	fan speed?



Data	Metadata
button mean	button pedestal
button RMS	button gain
trigger timestamp	quad offset
module location (e.g. 22E)	gain settings?
bunch current	timing settings?
	board temperature?
	board hardware revision?
	firmware version?
	fan speed?
	train #
	bunch #
	# of averaged turns

Button data/metadata

Data:

- ✗ timestamp, bunch current, location, button mean and RMS → one dedicated table

Metadata: write information to db table only when it is new

- ✗ button pedestal → one dedicated table
  - ➢ pedestals currently live in RD file, but have not been updated in a long time. New scheme could collect automated pedestal on Tuesday and populate db table
- ✗ button gain → one dedicated table
  - ➢ gains currently live in RD file, but have not been updated in a long time. New scheme would rely on periodic gain measurement/on a per-need basis (module swap)
- ✗ quad offset → one dedicated table
  - ➢ offsets currently live in dedicated text file and are updated on a per-need basis. New scheme could snapshot (w/ timestamp) text file to db when file changes.
- ✗ train # and bunch # → one dedicated table
  - ➢ more on this later

Button mean/RMS data type

Button mean and RMS information will be integer with a scale factor of 1,000 → save storage space and keep all the digits (and more) we might want. Reasoning:

- ✗ max # averaged turns: ~30,000 at 10 Hz
- ✗ smallest RMS noise is 8 ADU (electronics noise only)
- ✗ button mean standard error ≡ RMS noise/sqrt(# turns) = 8 /sqrt(30000)=0.045

It would be nice to have the button mean granularity up to 65,535.001 which requires a double-precision float (8 bytes), or a 4-byte signed integer using a 1,000 scale factor

Bunch current data type

Bunch current for stored beam can be as low as 0.01 mA and can be as high as 200 mA for all the trains/bunches summed together

PSQL smallint is a signed 2-byte integer allowing value up to 32,767

With scale factor of 100 for bunch current we can accommodate from 0.01 mA to 32,767 mA

Module configuration and metadata

Some metadata we might want to store in database is also information needed for configuring the module. Is the database purely for one-way downstream information storage or also a two-way for configuring the modules?

train #/bunch #: what is the flow of information?

- ✗ is the information first written/stored in database, then used down the line?
- ✗ is the information first used to configure, then written to the database as part of data collection?

Same questions apply to:

- ✗ # of averaged turns
- ✗ gain settings
- ✗ timing settings

More metadata

Do we want to monitor the following?

- ✗ board temperature
- ✗ board hardware revision
- ✗ firmware version
- ✗ fan speed

If yes, can we do it concurrently with data acquisition without interfering?

Interlocking from beam position

Assuming that it all happens inside the module for fastest response:

- ✗ “easy” route: linear beam position calculation in firmware (in FPGA)?
- ✗ “hard” route: non-linear beam position calculation in software?

`nonlin_bpm` and quad offset

We want CersV's `nonlin_bpm` to reconstruct the nonlinear beam positions

Currently, `nonlin_bpm` uses a hard coded quad offset file name and cannot deal with passing the file name as a configuration parameter

For CBPM3, quad offset information will live in database → plan is to call `nonlin_bpm` disabling the quad offset correction and apply it on the output of `nonlin_bpm` using the information from database

Additional materials

Add button RMS to short/long-term tables

Deal with timezone for snapshot and flush management

Data size/type test for short/long-term tables → estimate disk space needs and ensure we have access to it

Run stress test once all the above are settled

Create button pedestal cable and automated code to populate it

Create quad offset table and automated code to populate it

What to do with gain in the short term? All set to 1?

Create remaining metadata tables and scheme to populate them