A Web-based control and monitoring system for DAQ applications

Alexey Anisenkov (*BINP*)

Ivan Logashenko (*BINP*)

Daniil Zhadan (*BINP*)

CHEP 2018, Bulgaria, 9 July 2018

Outline

- The Role of monitoring in Online Computing/DAQ (Why do we need central monitoring tools?)
- Involved Experiments (CMD-3, Muon g-2, MRT)
- Architecture Overview (Web-based approach)
- Components of the system

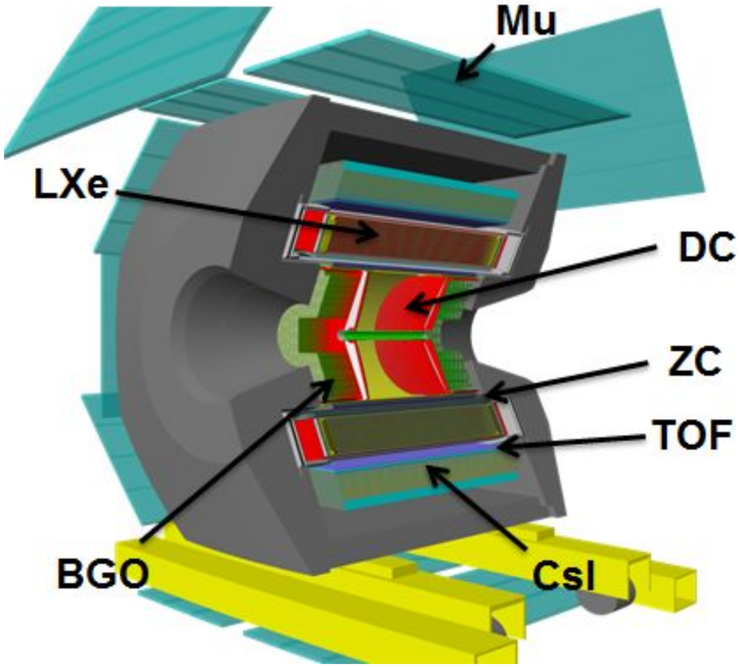
Role of monitoring tools in DAQ

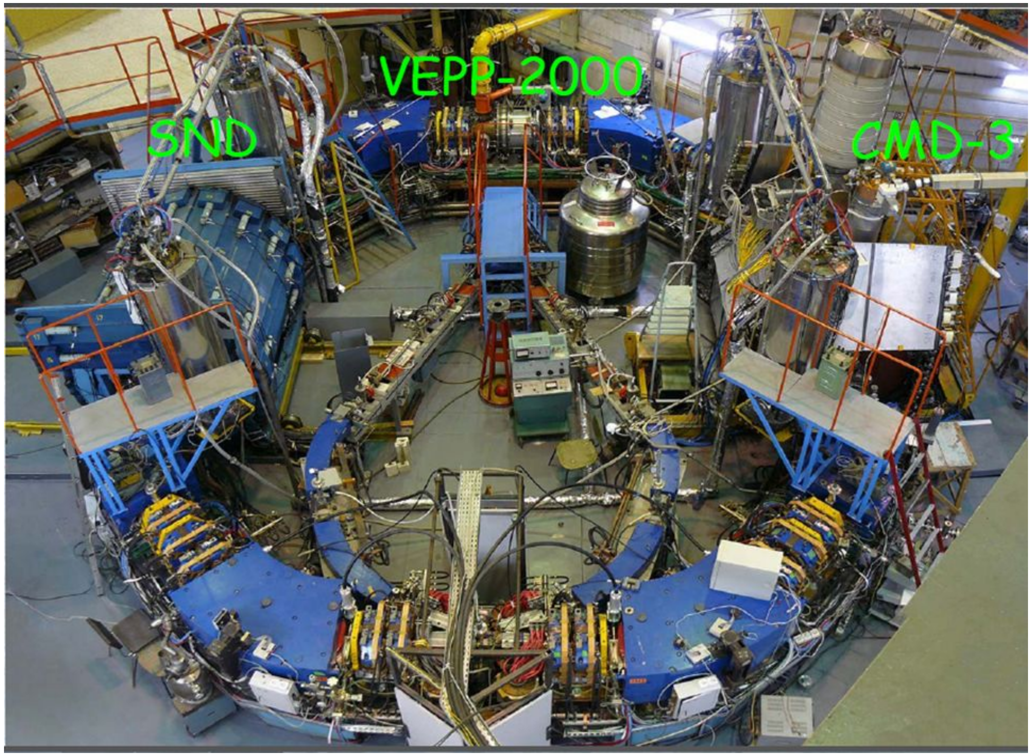
Slow Control and monitoring system is a vital part of any HEP experiment

- Monitor the status of DAQ and DAQ hardware
- Monitor physical and environmental conditions
- Control the quality of data taken
- Control and operate hardware equipments
- Guarantee safety and correct functioning of whole system

CMD-3 Experiment

The system discussed in the talk was developed for CMD-3 detector
Typical small-to-medium scale HEP experiment





- e+ e- collider VEPP-2000 at BINP (Novosibirsk)
- 7 detector's subsystems + cryo, gases, HV, LV
- ~ O(1000) environmental sensors
- ~ O(100) monitoring histos, data quality plots

- 60 authors
- 10k event size, 1kHz FLT rate

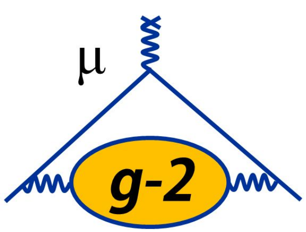
Basic considerations

Key requirements for the monitoring system:

- Independent of particular experiment
  (as much as possible)
- Modular structure
- web-based approach

Thanks to the modular approach,
parts of the system are used at two other experiments:

Fermilab



- Muon G-2 (250 authors)
- Larger than CMD-3,
  but same scale

BINP

- BINP MRT
  (X-ray tomography)
- Smaller,
  measurement station

Basic sources of monitoring data

During the operations DAQ and related systems produce a lot of information for experts and people on shift that need to be monitored and taken into account

Slow control

- Centralized Slow control hardware sensors
- Subsystems monitoring channels
- Archived Slow Control data

Offline Reconstruction

- Online Data quality metrics
- Slow Control software sensors
- Offline Data quality metrics

Online DAQ

- Direct read-out of front-end electronics (crates, subsystems)
- DAQ status metrics
- Run Log
- Nearline data processing

Key goals of high level monitoring system

We need a unified and user-friendly access to diverse pool of monitoring/control data:

- Access to real-time and archived data
- Different focus for shifters and experts
- Possibility to control detector subsystems
- Various helpers (data highlighting)
- Physicist should be able to extend the interface (min knowledge in programming)

Web-based approach meets well our goals

System architecture:
Why a web-based approach?

Modern Web technologies offer a big set of advantages and ready to use components out of the box.

Client-server architecture

- scalability and reliability
- extensibility (easy integration of experiment specific tools)
- hide direct dependency with front-end electronics and data sources

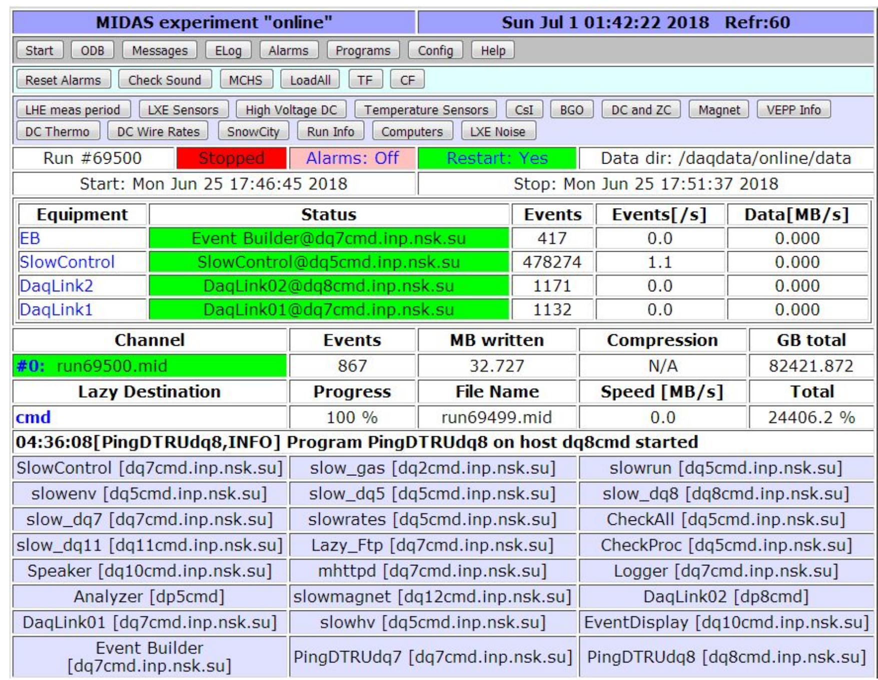
Web application

- cross platform compatibility (no dependency to client OS)
- accessible anywhere (can be even used remotely outside control room)
- cost effective and rapid development (thanks to Python, Django, and plenty of open-source web packages)
- easy customizable (CMS-like approach to edit pages)

Sidenote: MIDAS as core platform for DAQ & SC at CMD-3

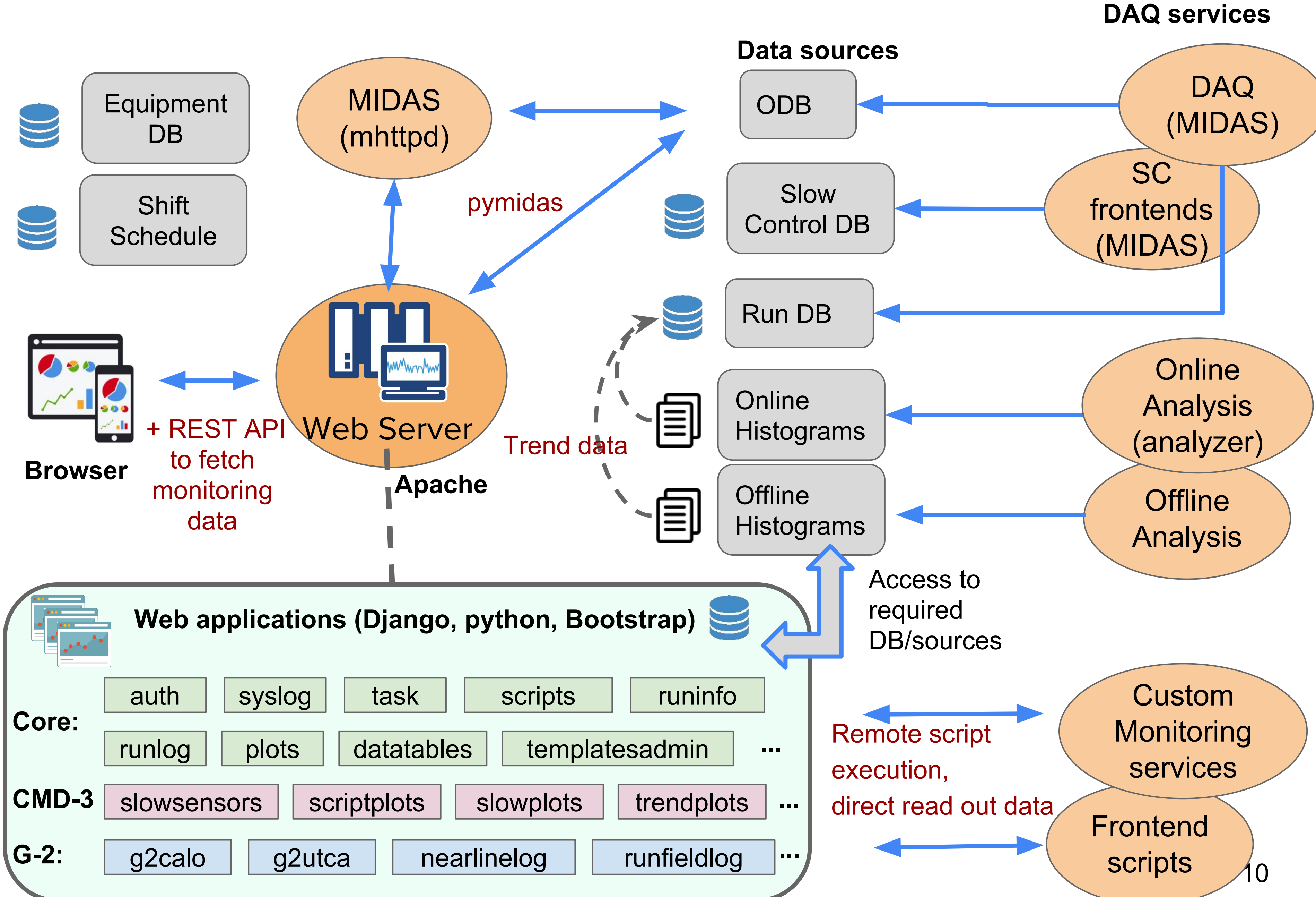
MIDAS is a rich data acquisition software developed at PSI and at TRIUMF

- Includes native Web Interface (mhttpd)
- Provides Online database (ODB) with tree-based structure
- Uses shared-memory Buffer for event collection and distribution
- Supports ROOT analyzers for online data monitoring (produces histograms)
- Frontend acquisition code written in C/C++



At CMD-3 we extended MIDAS API by implementing python library (pymidas) to access ODB and Buffer modules. PyMidas has allowed to apply easy integration with our DAQ services and in particular with web applications.

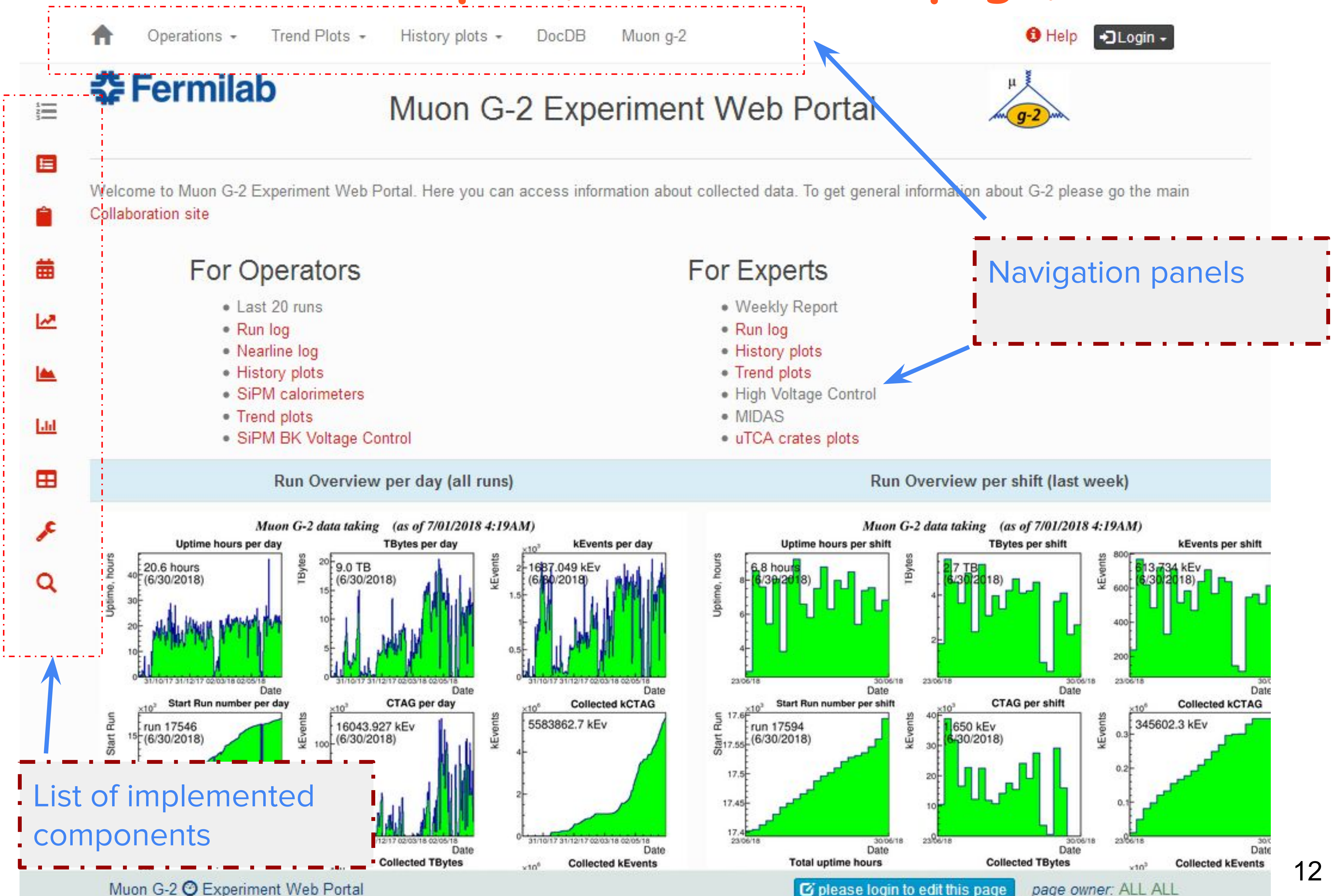
Architecture overview



Implementation details: Web2.0

- Apache/WSGI + Python + Django framework as server backend
- Independent database backends (PostgreSQL, MySQL, etc)
- Web Services technologies (REST API, WebUI, widgets)
- Bootstrap framework as HTML/CSS/JS client frontend (responsive, interactive, mobile-friendly)
- Client AJAX, JQuery plugins, own widgets, HTML5 vector graphics (datatables, treeview, calendar..)
- Plugin based approach (shareable applications in “core” re-used by many components)

Example (Main WebUI page)



Graphical component to draw plots

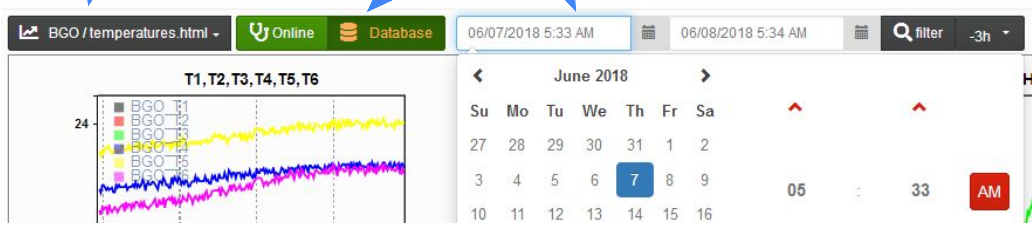
Own implementation of low-level plot.js widget based on D3.js

- Fully interactive, dynamic data visualization
- Data loading via REST JSON API
- Implemented as standalone JQuery plugin
- Draw several graphs on same pad within canvas
- Common X-axis slider for all plots on a page
- Predefined time windows
- And more..

Interactive plots: some features

Predefined plot presets

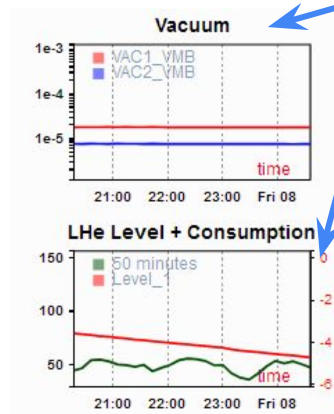
The same interface for real-time and historical data



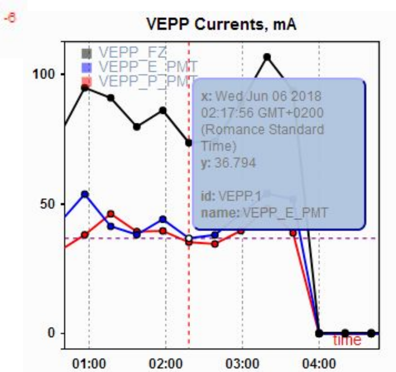
Automatic refresh for real time data

Ability to zoom in/out for x,y axis to get more detailed picture

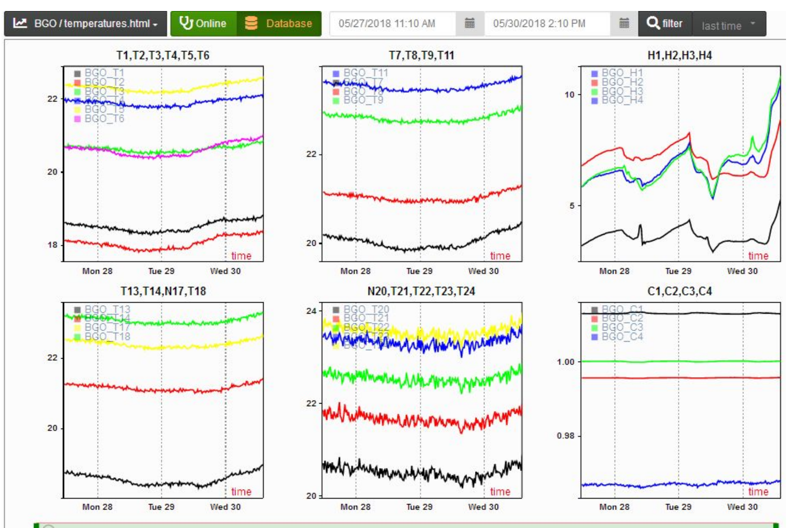
Log scale, 2 y-axis on same pad, custom data transformation (deriv)



Automatic zoom and switch from lines to points level depending on requested x-time window;
Point details pop-up window



Slow plots (time as x-axis)

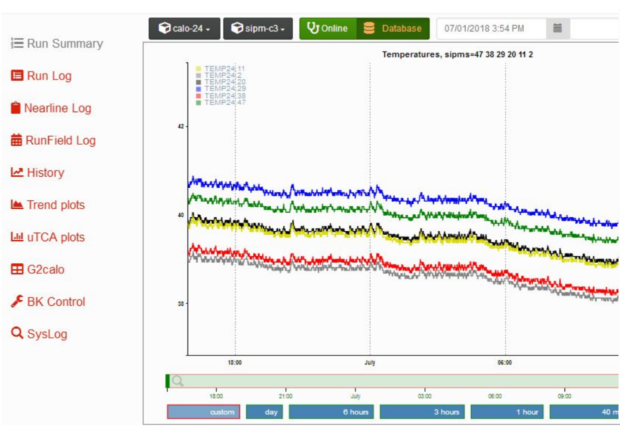


Graphical component: shared implementation

Given application is used as a base engine for following components:

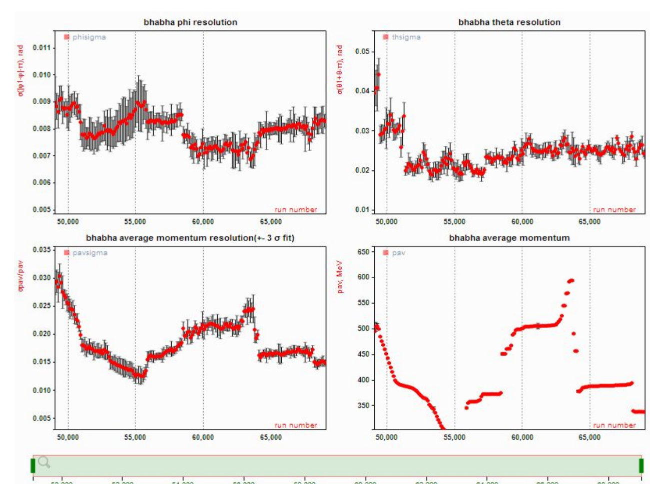
- Central Slow control data visualization (**slowplots**)
- Online and Nearline analysis data visualisation - run by run trending (**trendplots**)
- Custom data monitoring
  - Real-time read-out from frontend electronic (e.g. temperatures of SiPM calorimeters at G-2 - **g2calo**)
  - Draw monitoring data from custom db/source (e.g. monitoring of microTCA crate temperatures/params at G-2 - **g2utca** application)

G2calo plots



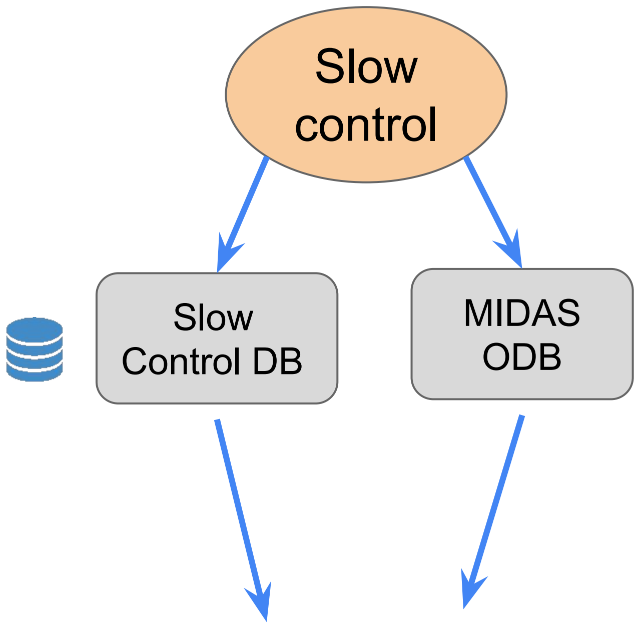
G2utca plots

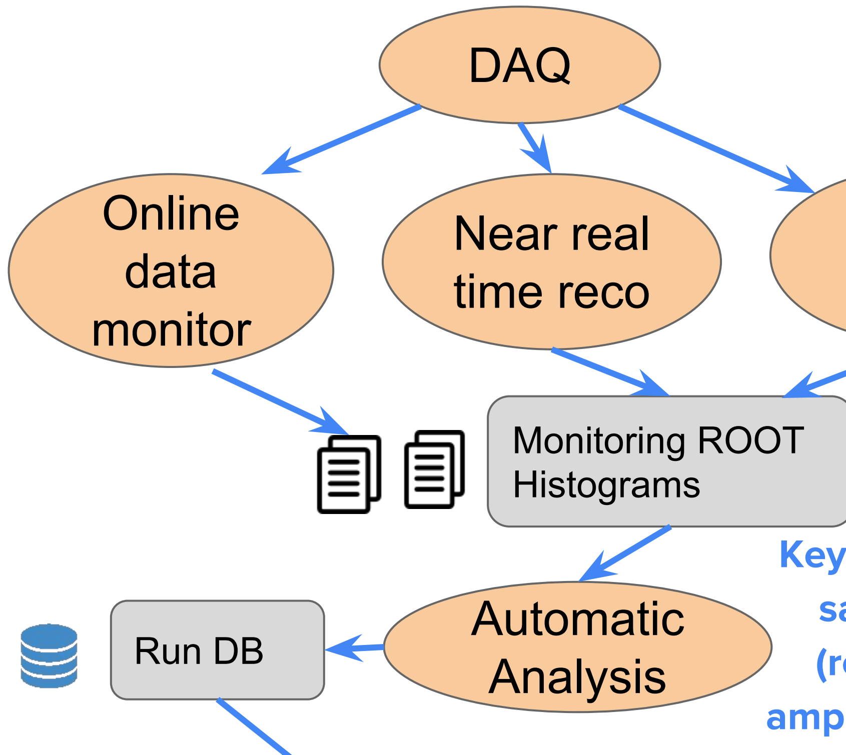
CMD-3 trend plots



Data quality plots (trend plots)

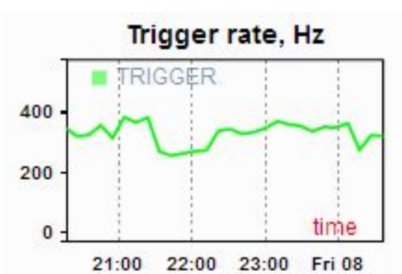
Different data flow to generate data quality metrics (online, nearline, offline)





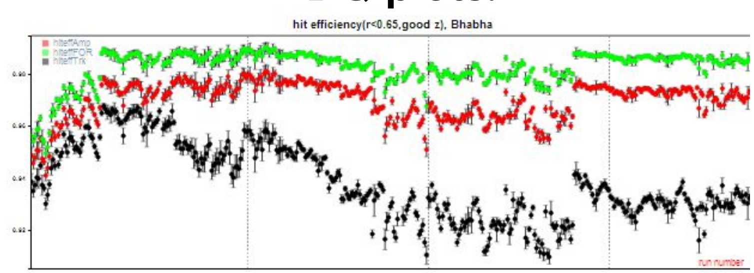
Key parameters are saved in RDMS (resolution, avg amplitudes, track rec efficiency, etc..)

Slow control plots:



Values vs time

DQ plots:



Values vs run #

Implementation feature: Django template tag as widget

Special template tag encapsulates all complicated logic and allows easy configuration of plots by users within WebUI

Edit Template: slowplots/presets/Run_Overview/Environment.ht

```
{% extends "slowplots/_base_preset.html" %}
{% load slowplot %}
{% load fields %}
{% block title %}Environment{% endblock %}

{% block main_content %}{{block.super}}

{% slowplot name='env' nx=2 ny=2 source="{{source}}"%}

source_url = {{declare.sources|get:source}}
refresh_cid = {{declare.refresh_cid}}

pad.0.title = Temperature, Celsius
pad.0.sensor.0.name = ENV.0

pad.1.title = Pressure, mm
pad.1.sensor.0.name = ENV.1

pad.2.title = Humidity, %
pad.2.sensor.0.name = ENV.2

{% endslowplot %}
{% load owner %}
{# keep the owner tag below to control who will be able to modify
this template #}
{% owner user='anisyonk' group='DAQ,ALL' %}

{% endblock %}
```

Backup file before saving? SAVE

We use Django tags to create "widgets"

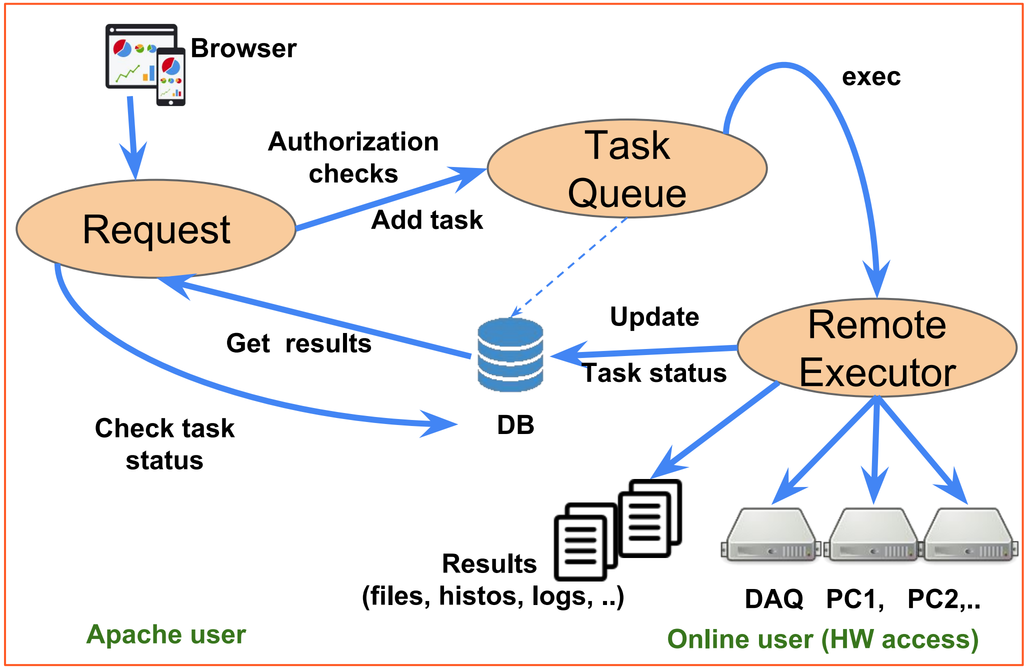
- Pages can be edited directly (thanks to **templatesadmin** app implemented)
- **slowplot** template tag specifies plot configuration (sensors, pads, colors, ranges, axis settings, transformations, auto zoom, etc..)

Remote script execution

The system is able to execute custom scripts from the web page, run them real-time at required DAQ machines, and report exit code/stderr/stdout back

Base scripts component:

- Use distributed task queue Celery + MySQL/RabbitMQ as message broker
- Register within the system corresponding Task and track its status in WebUI
- Use **template tags** approach to customize how data should be reported back to web
- Support for locking (multiple launch protection) + appropriate authorization checks



Typical use-cases and applications:

- Interactive hardware control (e.g. prepare boards for data taking, **runscripts** at CMD-3)
- To generate histograms/plots server-side with complicated analysis or involved several data sources using ROOT/JSROOT
(e.g. **scriptplots, offlineplots** at CMD-3, **trendplot** at G-2)

Runlog table view/operator helper (classic application)

Provides list of collected runs during shift with primary information exposed

- Interactive view to browse Run log table operated by MIDAS
- Ability to update Run details if need

Live filtering, customize columns, resolve runs by given shift/date

Provide shift overview in numbers

Complement Run details with parameters produced by Offline Analysis

Highlight bad Runs that require attention by operator

Links to online histograms and Run passport page

CMD-3 RunLog

Export | Columns | Filter | Reload | Show All entries | 10/5/2018 9:00 | Shift summary

Run	Q	Start time	Stop time	Events	Size	Shift	Copy	En MeV	Field Gs	Lum 1/nb	Offline Lum	Run comment
Filtered: 84 runs (65386 - 65469)		From: 2018-05-10 12:16:52	To: 2018-05-10 21:02:37	Collected: 18192148 ev	Collected: 156.84 GB					Coll: 368.066	Coll: 346.054	Uptime: 8.47 hours (96.64%)
65469	Y	2018-05-10 20:57:13	2018-05-10 21:02:37	214203	1.88 GB	Fedotovich Bachtovoy	HDD	390.5	13000	3.178	2.926	Standard trigger. HV 2110 V DC. 8 CsI chan. excluded from trigger summ
65468	Y	2018-05-10 20:51:00	2018-05-10 20:57:05	214919	1.88 GB	Fedotovich Bachtovoy	HDD	390.5	13000	4.168	3.867	Standard trigger. HV 2110 V DC. 8 CsI chan. excluded from trigger summ
65467	Y	2018-05-10 20:45:23	2018-05-10 20:50:52	227781	1.88 GB	Fedotovich Bachtovoy	HDD	390.5	13000	3.362	3.138	Standard trigger. HV 2110 V DC. 8 CsI chan. excluded from trigger summ
65466	Y	2018-05-10 20:39:43	2018-05-10 20:45:15	216364	1.89 GB	Fedotovich Bachtovoy	HDD	390.5	13000	4.784	4.516	Standard trigger. HV 2110 V DC. 8 CsI chan. excluded from trigger summ
65465	Y	2018-05-10 20:34:47	2018-05-10 20:39:35	214773	1.88 GB	Fedotovich	HDD	390.5	13000	4.589	4.341	Standard trigger. HV 2110 V

Run Field Log at G-2

Run	Start time	Stop time	Quality	Field (kHz)	PSFB Curr. (mA)	Trolley pos	Trolley mode	Flux	PP	PS FB	PS Active	SCC	Trolley	Galil	FP	Run comment
Filtered: 50 runs (5267 - 5317)	From: 2018-06-26 07:16:44	To: 2018-06-30 10:47:15														Uptime: 90.08 hours (90.53%)
5317	2018-06-30 09:03:32	2018-06-30 10:47:15	Y	48.48	0	175	Idle	✔	✖	✔	✖	✔	✖	✖	✖	Magnet ramping down
5316	2018-06-30 08:26:27	2018-06-30 09:01:43	Y	48.52	0	175	Idle	✔	✖	✔	✖	✔	✖	✖	✔	Field run after magnet recovery
5315	2018-06-30 06:24:13	2018-06-30 08:26:14	Y	52.03	13.5	175	Idle	✔	✖	✔	✔	✔	✖	✖	✔	Field run after magnet recovery

Other components

Not covered in this talk

- Real-time monitoring using table representation (**slowsensors**)
- Overall information about Runs (**runinfo**)
- Update forms to change various information in databases
- Changes log and history of user actions made within the system (**syslog**)
- Custom applications for particular subsytems:
  - hardware control modules
  - interactive forms to configure boards (e.g. **triggersettings**)
  - remote execution of chain of scripts (**loadelectronics**)

Conclusion

Modern Web 2.0 technologies and open source tools can be effectively used to build functional, handy and attractive applications for Slow Control and monitoring system

- The CMD-3 web-based monitoring system provides full access to whole set of monitoring and control data as well as possibility to configure hardware equipment
- Thanks to modular approach and experiment-independent architecture, parts of the system are also used for other experiments (Muon G-2, BINP MRT)

Thank you for your attention!

Script plots example

Run custom analyzer (python ROOT script) server-side to build plots/histograms

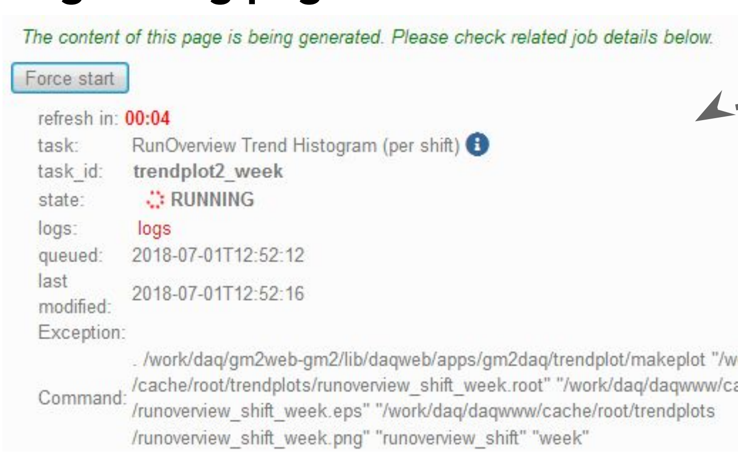
- Cache results (pictures, root, logs, eps)
- User can implement own ROOT script
- Once a script is uploaded to the server the integration into any web page is just one line using special **template tag**
- Additionally use **JSROOT** to interactively browse ROOT files content

Template tag to visualize script result

```
{% trendplot name="runoverview_shift"
query="week" redirect="reload"
thumburl="trendplot-info" width='500'
cache_time='4h' force='1' %}
```

Rebuild button

Progressing page



Result (Run Overview per shift)