Online Software Application Configuration and Deployment

Brett Viren

Physics Department

BROOKHAVEN NATIONAL LABORATORY

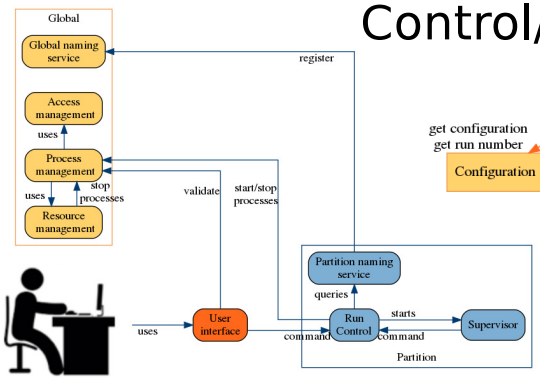
EVENT – DATE

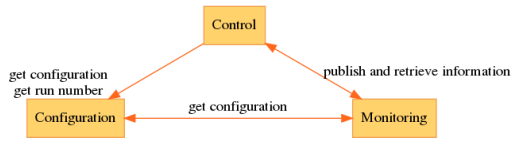
Outline

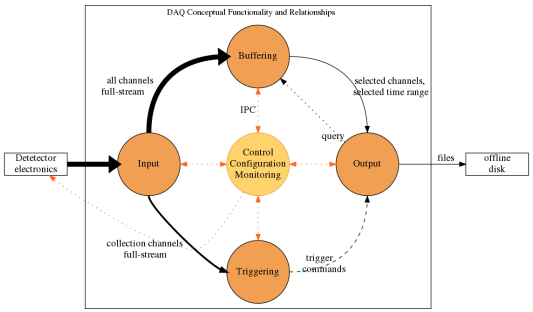
Caveat: I'm not sure what I'm supposed to say!
So, I'll just give some ideas and discussion starters.

Current DAQ Subsystem Concepts

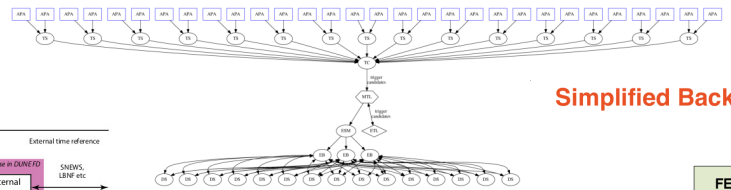
Control/Config/Monitor





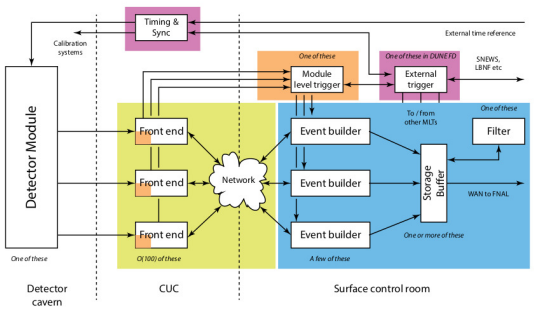


Data selection



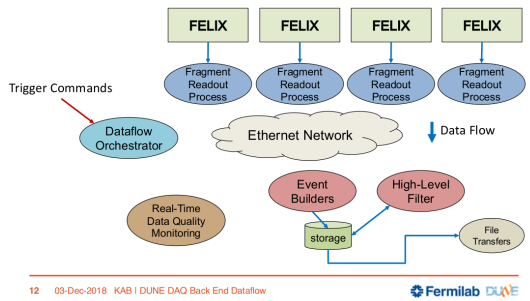
Readout

The Baseline Design



- Plus: CCM; Online Tools; Infrastructure and Integration

Simplified Back End Dataflow Diagram



12 03-Dec-2018 KAB | DUNE DAQ Back End Dataflow Fermilab DUNE

Incomplete List Of DAQ Apps

CCM run number service, various configuration, partition and other manager services, RC itself, expert systems and human UIs, log aggregators, metric monitors, configuration sequencing, detector electronics config service(?).

FE TP source, buffer interface service.

DS TC processors, MLTs, ELTs.

BE Data flow orchestrators, event builders.

DQM Suite of "live" data quality processing and visualizations.

ad-hoc Detector debugging, data stream taps, one-off monitoring.

post-hoc Nearline processing / data selection.

Categories of Configuration

initial initial parameters given at execution time to **DAQ applications**, not tied to run changes.

dynamic (re)configuration of **DAQ applications** at subrun[1] boundaries.

services "`/etc`" files for any supporting services (SSHd, PTP, Supervisord, Ganglia, CVMFS, Apache, RDBMS, ...).

OS definition of base set of packages and their versions, Linux boot/tuning parameters.

[1] Data Model definition

Some questions we must answer

- How do we manage the parameter set(s), deploy them and schedule, maintain history and rollback those deployments?
- How do we handle interdependencies? eg a subrun change may require new DAQ applications to execute.
- What is **definitive source** of configuration information and what are **derivatives** and how to go from source to target?

Jsonnet Based Configuration Management

- Jsonnet is a sane JSON-like functional+OO, data structure language.
- Established language, used to configure Kubernetes and Wire-Cell Toolkit
  - WCT can provide Jsonnet code for complex graph construction.
  - Jsonnet syntax is simple and somewhat FHiCL'esque.
- Directly supports compiling to JSON, INI, YAML
  - easily generate arbitrary formats with JSON+Jinja2 templates
    eg as used in `dune-tdr/dune-params`.
- A single, central Jsonnet file set could build out to forms taken by all configuration consumers.
- Encourages Don't-Repeat-Yourself patterns, consistency as well as factoring of configuration effort/duties.
- Could maintain Jsonnet source in Git for history, branching, rollback.
- The approach decouples the development and the consuming of configuration information, helps when debugging and developing apps.

The model: the git repo represents all past and possible future configurations. The point in that commit history actually applied at any time is determined and recorded by CCM.

Application Deployment Scope

- Assume Computing provides/manages OS and "some" services. DAQ+Computing must formalize the division.
- DAQ application software binaries must be available on DAQ host computers.
- Expect to need multiple versions available for execution and to promptly add new ones.
- Must determine mechanisms to execute, monitor and reap executing apps and to define, construct, debug, monitor and archive their execution environments.

Some Deployment Technologies to Evaluate

Ansible central configuration files pushed to hosts over SSH, services restarted, can reboot, act across host categories or ad-hoc/targeted.

Puppet like Ansible, but hosts run daemon that polls master.

Supervisord daemon to run other applications, can respond to events and be monitored via XML-RPC, batteries mostly included but also extensible via Python.

Containers bake OS or OS+apps into image, run multiple containers on a host.

Kubernetes large scale container orchestration.

Launching and Reaping DAQ Applications

Normal starts, "self healing" and "zero-downtime" require:

- Launching a large and varied set of DAQ applications.
- Specifying their operating environment.
- Matching apps to hosts given **constraints**.
- Monitoring their execution and detect crash (IPC presence).
- Potentially reaping errant instances.
- Minimizing delay between launch request and application execution.

It occurs to me this closely resembles ATLAS's PanDA or other "pilot based" batch system.

No tight coupling

- Don't have a major DAQ application (eg, RC) talk directly to a database.
- Have RC **PUB**lish information and another app **SUB**scribe in order to fill a database.
- Have RC query a configuration application, which may front a DB or other source of configuration (eg, Jsonnet).

Example: Access FE Buffer

```
10s RAM buffer <--> BIS <--> BR
               <-->debugging (``oscilloscope'')
               <-->Ar39 special "patch" readout
```

Who can access what and how?

- Disallow any human to log in to a production account.
  - Eg: take model how Apache, PostgreSQL, etc is run on most OSes.
- Limited number of expert accounts on DAQ hosts.
  - SSH key access control.
  - Used for testing of new versions, debugging, ad-hoc application execution.

FIN

Reminder: Zero-downtime Reconfiguration Mechanism

1. RC decides to change a subrun some time in the near future.
2. Determine target **data time** for it to apply.
3. Any newly required DAQ apps are started via RC command.
4. RC forms reconfiguration commands with **data time** stamp.
5. Send reconfig commands to all DAQ applications (old and new).
6. Both old/new apps immediately form any new connections, keeping existing ones nominal.
7. New apps accept input to monitor for **data time** but otherwise idle.
8. Old apps operate nominally.
9. When **data time** is reached, apps finalize execution of reconfiguration command, flush buffers, drop any unwanted connections and terminate if instructed..