Software and Computing for DUNE CE Testing

Brett Viren

Physics Department

BROOKHAVEN
NATIONAL LABORATORY

CE WS – July 2018

Outline

Overview

Configuration Management

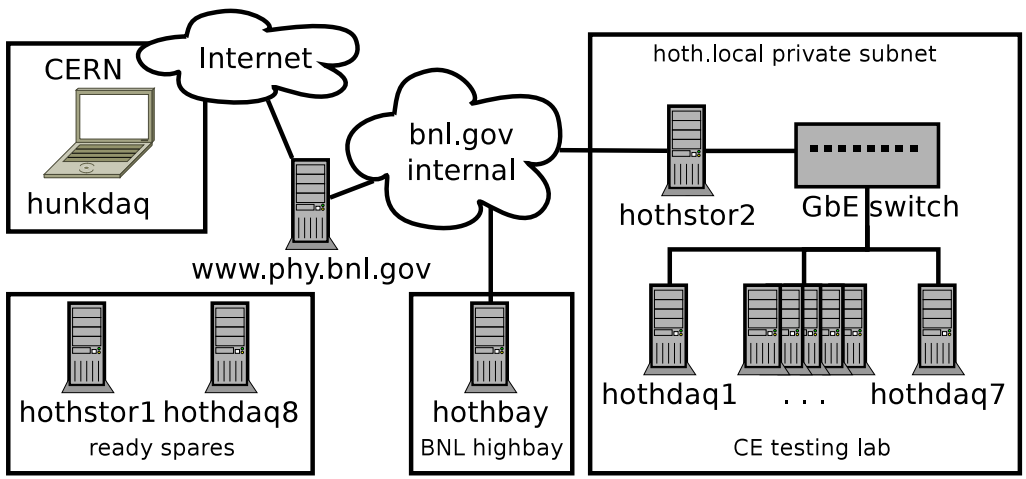
Technical Policies

Computing Hardware

Software

protoDUNE CE Testing S&C

- 2 servers, 10 DAQ hosts (one as a laptop to CERN).
- Semi-automated S/W release, build and deployment.
- Centrally managed, automated configuration management.
- Data provenance, result summaries, and computer monitoring systems.

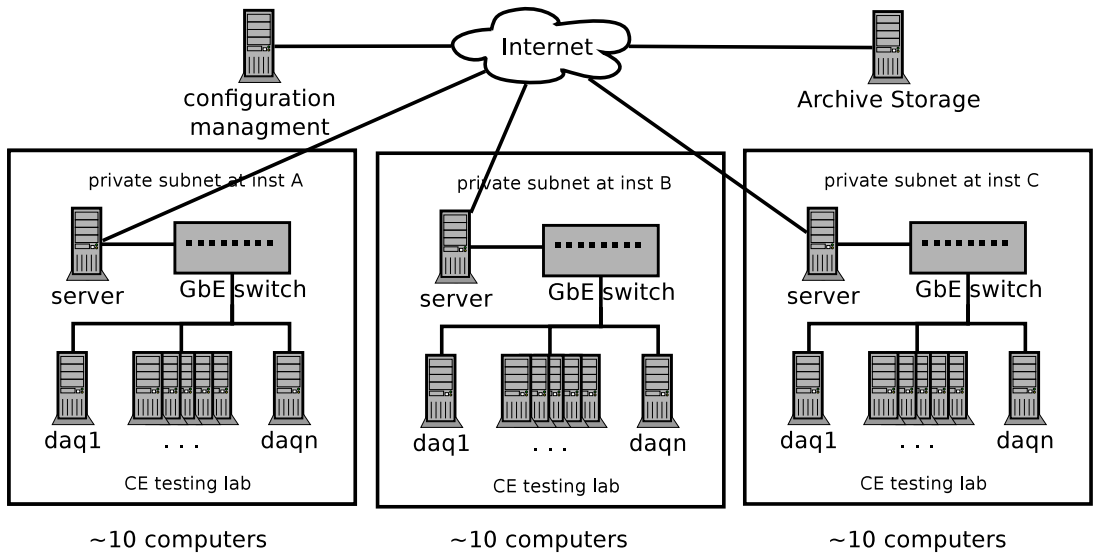


Based on this experience I'll try to suggest how to scale to DUNE.

Assumptions toward DUNE

DUNE will test CE similarly to how protoDUNE did, except

- DUNE likely must test fewer units per APA.
- But DUNE has $> 30\times$ more APAs.
  - → So, still must **scale up** total units and respond by **scaling out** testing to more **more institutions**.
  - → Envision an integrated but distributed system:



Overview

Configuration Management

Technical Policies

Computing Hardware

Software

Configuration Management System

Why do we need a configuration management "**system**"?
Why not just write a "how to setup a DAQ" document?

- **Consistent and known** configurations must be applied across $\approx$ 10 DAQ hosts + a few servers $\times$ several institutions.
- **Changes will happen** and need to be **tested, accepted and propagated**.
- **Configuration is tedious and time consuming** and **automation** is a force multiplier, conserves limited effort (and sanity).
- **Testing can be chaotic** and enforcing some structure reigns that in.
- **CE Testing workers have enough problems**, configuring the computing system should be the least of their worries
  - $\rightarrow$ but they **must have some control** of their computing!

CM Lessons from protoDUNE CE Testing

- protoDUNE CE testing used **Ansible** for CM, worked well.
  - → spin up a new DAQ host in minutes.
  - → often easier to use Ansible than directly edit target config files.
  - → it should be easy to scale to more stations and institutions.
- **Must be strict** in order to keep the CM system authoritative but also **must be responsive** to requests especially in face of emergency time crunches.
  - → **progress before perfection**, a few experts need `root` access and be empowered to subvert CM in times of crisis and at least temporarily.
- Besides Ansible, freedom to choose OS (Ubuntu) and other implementation details made this feasible with limited available effort.
  - → Very important for those experts that will actually develop, use and maintain the systems to define what the system will look like.

Overview

Configuration Management

Technical Policies

Computing Hardware

Software

System

- All DAQ hosts live on a NAT'ed subnet not directly accessed from the internet nor the institutional LAN.
  - Remote access via a bastion/gateway bridge-host
- DAQ hosts require only services available on the subnet for normal, production operation.
- All DAQ hosts are independent from each other.
- DAQ hosts have minimal dependency on servers: eg, local accounts, no network FS.
- Control remote access to DAQ hosts and servers via SSH keys managed by CM.

Users and Access

A starting point:

- Strict user roles:
  - `oper` production data taking, owns all production data
  - `inst` installs software, owns all software files
  - `arch` archives data, owns any copies
  - users DAQ s/w dev, write data only in scratch disk areas.
- Only `oper` logins to local DAQ host console/desktop
  - Only account with a password, not useful for remote access.
  - All other users: remote access via SSH.
  - This includes Ansible which also needs `root` access.
  - Access to `oper`, `inst` and `arch` only for Ansible + a few experts.

Data

- Bulk of disks are reserved for production testing results.
- A testing *job* consumes $\mathcal{N}$ *results*, produces 1 *result*.
- A *result* is a directory w/fixed patterned name of:
  - the **type** of unit tested (FE, ADC, FEMB, etc)
  - the test *application* **name**
  - the starting **time** of the test to the second
- A *result* directory holds:
  - A `params.json` with all input parameters for the *job*.
  - A `summary.json` with an app-specific summary of its *result*.
  - Additional sub directories and files following a per-app schema.
- A given *application* may strictly acquire data ($\mathcal{N} = 0$) or consume existing *results* in order to produce its own.

Develop DAQ s/w layers and modules to enable/enforce this.

Overview

Configuration Management

Technical Policies

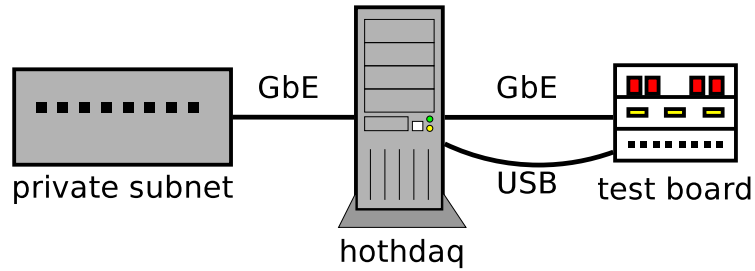
Computing Hardware

Software

Computing Hardware

Low h/w barrier to become a testing site.
Commodity, mid-range Linux PCs sufficient.
No need to dictate detailed specs.

But h/w homogeneity at a site is a good thing.



At least 1 server PC per testing lab:

- Bastion SSH gateway / NAT router for the private DAQ subnet.
- Provide network services: DHCP, NTP, HTTP, Ganglia.
- Also: database (if used - needs discussion), s/w build host, off-DAQ archive storage (and/or just send it all to FNAL asap).

of DAQ PCs depend on scope/role of each testing lab:

- At least 2× GbE NICs: network + boards access
  - Common NIC h/w model nice to keep consistent device name.
- Average CPU (i5) and RAM (8-16 GB), kbd/mouse/monitor.
- Data disks sized based on expected test roles, $\mathcal{O}$(10 TB).

Also: GbE switch, various CAT 5e/6 patch cables (LN2 tends to destroy them).

Overview

Configuration Management

Technical Policies

Computing Hardware

Software

protoDUNE CE Testing Software

The protoDUNE CE Testing software was rather successful but some more effort is needed to extend it to DUNE.

+ Source in `git` on GitHub.
+ Managed, tagged releases, automated build and deployment.
+ Per-DAQ host version control with rollback.
+ Python-based, low barrier to contribution, somewhat modular, somewhat layered.
+ Some abstractions developed to cover some common aspects.
- DAQ code quality can be improved: avoid copy-paste, globals, multiple solutions to common problems, other anti patterns.
- Summary web pages were a quick hack that shouldn't be carried over.
- Data provenance system (Sumatra) wasn't quite ready and should be reevaluated.

The needed improvements are easily achievable by a small team. Eg, less effort that will be needed to adapt to new CE hardware.

DUNE CE Testing DAQ S/W Recommendations

- Stick with Python, limit use of ROOT but use it via PyROOT if required.
- Make a *job* an object, not an executable, avoid `subprocess` and `exec`.
- Factor jobs into smaller parts to allow reuse
- Develop job sequencer to support pipelines/graphs of jobs.
- Define and abstract all required protocol behavior (eg, consumption of config, creation of result directory, production of `params.json` and `summary.json`).
- Provide standard file I/O functions for common data types.
- Module-first design, top-level CLI/GUI code should be almost empty.
- If DAQ GUIs are to be used, avoid extensive hand crafting, abstract commonalities into modules/classes/functions.

Databases and Summaries

- protoDUNE CE Testing used file system of result directories as primary database.
  - I think this mode CAN scale to multiple institution.
- protoDUNE CE Testing Summary Web Pages produced via a static site generator with access to the **summary.json** type files from all results.
  - Static site generators backed by a build system is easy and powerful.
  - It needs a way to aggregate **summary.json** type files.
  - The exact hand-crafted system probably should NOT be extended for DUNE.
- At any time we can upload protoDUNE CE testing summaries to a RDBMS type database.
  - DUNE should probably plan to do this as SOP.

Summary

- protoDUNE CE Testing software and computing was pretty successful.
- We MUST extend to multiple institutions for DUNE.
  - Need to take care to manage this extra complexity while still giving sufficient **control to local experts and workers** while also **taking away burdens**.
- Some of protoDUNE CE Testing S&C can be directly extended:
  - Ansible Configuration Management system
- Some deserves some upgrading:
  - DAQ Python toolkit
- Some needs reimplementation and new development:
  - Result summary web pages
  - More formal RDBMS summary and provenance database

$\mathcal{FIN}$