Continuous Deployment Automation in Supercomputer Operations
Techniques, Experiences and Challenges

Matteo Chesi, Nicholas P. Cardo, Miguel Gila
May 9th, 2019
Motivations
Manage more resources with less people

- Model: Cray XC40/XC50
- Interconnect: Cray Aries (dragonfly)
- 5320 XC50 Compute Nodes
  - 2 x Intel® E5-2690 v3 @ 2.60GHz
  - 1 x NVIDIA® Tesla P100 16GB
  - 64 GB RAM
- 1813 XC40 Compute Nodes
  - 2 x Intel® Xeon® E5-2695 v4 @ 2.10GHz
  - 64/128 GB RAM
Cray CLE specific motivation

“Cannot apply changes to Programming Environment on nodes with running jobs”

- Issue introduced with the upgrade from CLE6.0-UP04 to UP07 (CAST-19800)
  - PE deployment changed default from “tree” to “squashfs” images
  - PE on elogin nodes is supported only with squashfs images (CAST-20346)

- **PRO:** squashfs images are better for DVS as single files to distribute

- **CON:** Need to run ansible to apply a change to the PE with squashfs:
  - Ansible unmounts previous image, mounts new image and it syncs data between the new image and the O.S. (i.e. modulefiles)
  - It causes a **disruption** of the environment of running jobs
  - We used to apply changes to PE on systems live
Forward Looking

Adoption of new IT Operations best practices

“What if we can just focus on the nature of changes and do not worry about their deployment process?”

Continuous Deployment
Cloud vs HPC

- standard hardware
- on demand computing
- idle resources

VS

- specialized hardware
- batch scheduled computing
- ~100% busy resources

"Can we automate change deployment on a production HPC platform?"
Work
Continuous Deployment Process - Design

- **Objectives**
  - Ease of use
  - Reproducability for subsequent updates
  - Reduction of required manpower

- **Foundation Work**
  - Reframe
  - Cksys Tests
  - Node LifeCycle Management (NLM)
Node Update

“Newly available update release could be handled as a special case of node failure.”

Design Elements:
- Exclude Node Reboots
- Idle Nodes
- Scalability
- Publish and Retire Updates
- Node Selectability
- Restrict Simultaneous Updates
- Implied Boot State
- Fully Automated
- Single Reservation

# ./manage_node_updates --help
This script manages node-update tasks on the system.

manage_node_updates <command> <options>

Possible commands:
* list
* create <update_id>
* delete <update_id>
* enable <update_id> <nids|all>
* disable <update_id>
* rename <update_id> <new_update_id>
* help
Node Update - Workflow

1. **AVAILABLE PRODUCTION-READY NODE SET**
2. **EPILOG NODE UPDATE CHECK HOOK**
   - need update
3. **JOB**
4. **ARE THERE ANY FREE SLOTS IN THE RESERVATION?**
   - found candidate
5. **IDLE NODES UPDATE CHECK CRON**
6. **ARE THERE ANY FREE SLOTS IN THE RESERVATION?**
   - no candidate
   - cron check
7. **READY**
8. **RUN UPDATE**
9. **UPDATING**
10. **UPDATE END + CKSYS**
11. **DONE**
12. **FAIL**
13. **EXTERNAL MONITORING**

Node Update reservation (size limited)

- no update
- release
Node Update – State Diagram

- READY
- UPDATING
- DONE
- FAIL

States and transitions:
- IDLE: update available → node reserved
- UPDATING: updates started
- DONE: remove from reservation
- FAIL: issues
- SUCCESS: remove from reservation
Node Update – Orchestrator

- No single daemon can handle correctly simultaneous RPCs \( \Rightarrow \infty \)
- Slurm control daemon is no exception
- NU workflow implies that nodes ask how many slots are free in the reservation:
  
  scontrol show res=nodes_update

- When a large job completes, a large number of nodes should execute this command simultaneously, but only few of them should be granted to run.

**Orchestrator**

- is the only agent entitled to communicate directly with the slurmctld
- aggregates compute nodes requests and cache slurmctld answers
- Communications between compute nodes and the orchestrator happen via a message bus (provided by Apache Kafka®)
- is the manager of nodes_update reservation
- schedules tasks on compute nodes for NU process
- is meant to become a distributed agent at higher scales
Results
Results and Next Steps

- **Node Update** is currently developed on Piz Dom (64 nodes), TDS for Piz Daint.

- **Staged Production Rollout:**
  - **First Stage:** deploy on Grand Tavé, KNL System (164 compute nodes)
  - **Second Stage:** deploy on Piz Daint (~7000 compute nodes) in dry-run mode
  - **Third Stage:** turn off dry-run.

- Development work will continue to fix issues and add features that each stage will highlight.
Future looking

- Enhance scalability
- Software Abstraction Layers: Performance vs Flexibility

- Expect contributions from providing:
  - manageability of all the hardware and software components/layers
  - more robust HSN
  - quicker reboots
  - closer collaboration with customers
Thank you for your attention