Los Alamos NATIONAL LABORATORY

Zero Downtime System Upgrade Strategy

Alden Stradling, Joshi Fullop

May 9, 2024

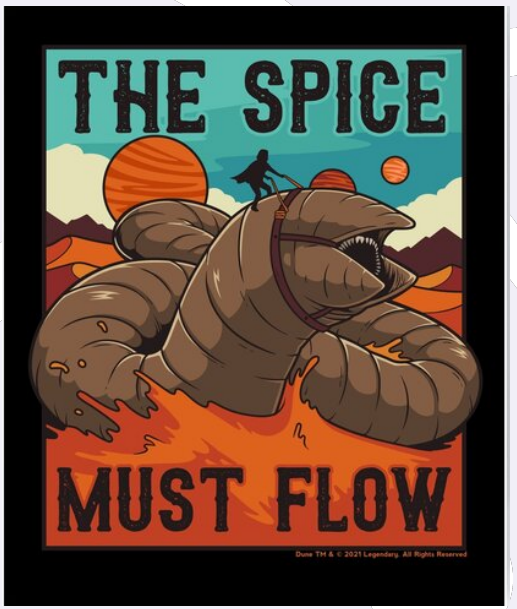
LA-UR-24-24407

Toward Zero Downtime

- Clarifying:
  - Zero Downtime means that a given cluster is never 100% unavailable to users at any given time.
    - There's always some loss of efficiency during an upgrade
  - This focuses on Cray EX with CSM > 1.3.X
  - Will not work for all use cases, depending on appetite for risk.
    - Major Slurm upgrades may be better done with a downtime
    - Major fabric upgrades will require downtime

Just Trying to Keep My Customers Satisfied

- *The science must flow*. But systems must be patched.
  - Visible system downtime is… unwelcome
  - System flaws introduced adding desired changes are… even more unwelcome
  - Visible downtime that reverts the desired changes that introduced the system flaws is… supremely unwelcome
  - Visible downtime to finally reintroduce the desired changes in a way that works is… grudgingly accepted
- Ideally… what if users got the changes and didn't have to get the downtime at all?

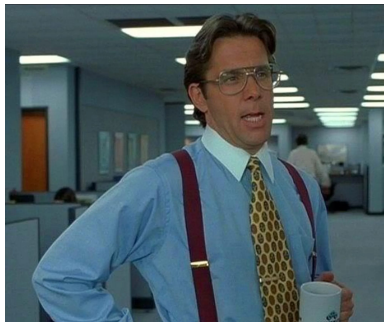


Efficiency Comparison (ex: 6,000 nodes, 8h downtime)

- Classic downtime:
  - Node Hours (NHrs) = Nodes down * hours down
    - 6000 N * 8h = **48,000 NHrs**
  - What if you hit a roadblock and hold overnight?
    - 6000 N * 28h = **168,000 NHrs**
- Rolling downtime:
  - Prep time:
    - **168 NHrs** per week per node. Easy to take 1 node for initial phase of rollout
    - Take 16 nodes: **2,688 NHrs/week**. Very moderate by comparison.

Efficiency Comparison (ex: 6,000 nodes, 8h downtime)

- Rolling downtime (continued):
  - Admin scale testing — say we take 10% of the cluster for a test
    - Break-even is at 80h, or ~3 24h days (far more than usually needed)
      - 4h at 600 nodes = **2,400 NHrs**
    - Distinct from user testing
  - Rolling reboot at the end of testing:
    - 90% of the cluster at 30m per rebooted node = **2,700 NHrs**
    - **(2,700 + 2,400 + 168) NHrs = 5,268 NHrs << 48,000 NHrs**



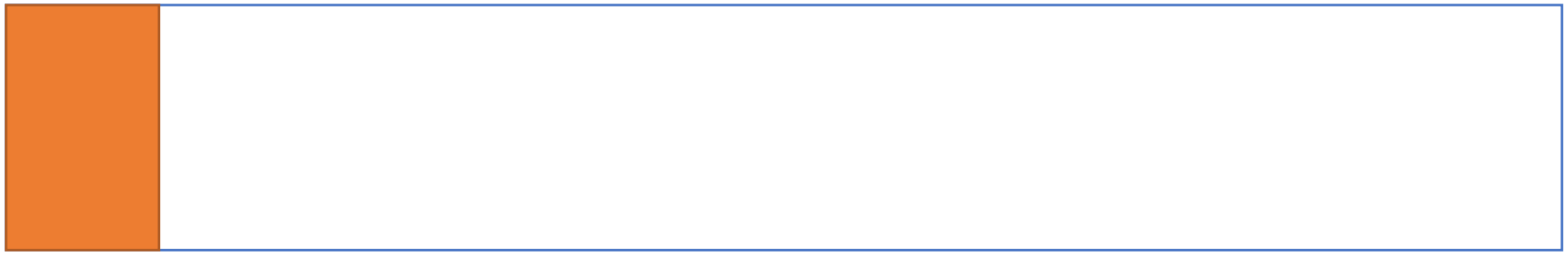
That would be great.

Further Reading

- This approach is not just system administrator's intuition. It is supported by workflow traces and simulations.
  - "Incorporating Staggered Planned Maintenance Reservations to Improve Performance in Computational Clusters". CLUSTER Workshops 2023: 32-36
    - See: https://dblp.org/rec/conf/cluster/JonesWHGDS23.html
- Using the BatSim to simulate realistic workload behaviors
  - Binned rolling upgrades demonstrate significant improvements in queue wait times

Classic Downtime

Rolling Reboot Outage



What Prevents This Utopia?

- **Risk!**
  - We take the system away because if the changes screw up production jobs, there's a real impact
  - Changes in config and deployed software create divergences in results. Users need to be clear on the running config
- **Inertia!**
  - Familiarity leads to speed. New approaches slow things down initially.
- **Complexity!**
  - More moving parts in an upgrade when allowing for user job continuity. Needs to be automated to avoid admin error and delays

De-Risking, Breaking Inertia, Encapsulating Complexity

- Prerequisites:
  - Automated Workload Management (WLM) reconfiguration (complexity)
    - Slurm in this case, but by no means exclusive
  - Simple tooling interface (inertia, complexity)
  - User front-end node channeling (de-risking)
  - Robust against in-flight failure (de-risking, complexity)
  - Low-friction rollback (de-risking, complexity)
  - Linking operations with image and config identifiers (complexity)
  - WLM config file state management with VCS (de-risking)

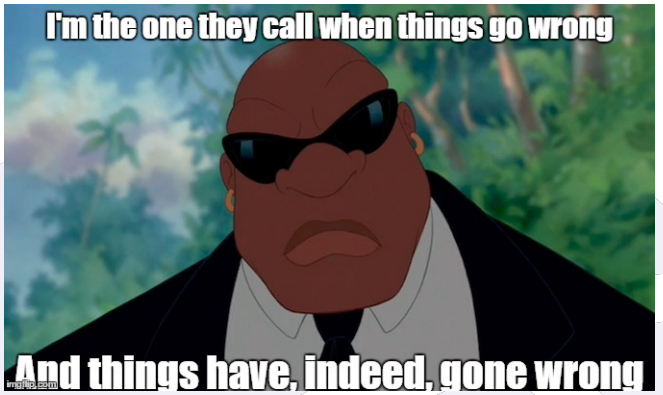
Getting It Done: Existing Tooling

- Using existing LANL WLM node list creation scripting
- Using existing Slurm tools `SlurmctldParameters` and `RebootProgram`
- Using existing LANL WLM scripts for responding to `RebootProgram`
- Using existing BOS v2 in conjunction with `RebootProgram`
- Bringing them together with relatively simple scripting:
  - Inputs are BOS template and node count to roll out
  - Keeps state in rewritten `slurm.conf` file, stored in Git repo periodically
    - Propagated with configless Slurm to avoid a lot of node work
  - Hits preset limits (1 node, small test, scale test, rollout, rollback)

Just Trying to Keep My Customers Satisfied

- User interaction:
  - Users use clusters as normal
  - Users who want preview access to the new config (testing, builds)
    - Log into a FE designated and tagged with the new build, DNS name
    - Work with nodes tagged with the appropriate Slurm `Feature`
  - When the upgrade is ready to roll out wide:
    - Communicate that the cluster will be moving over
    - Reboot all but one remaining FE, lock submission from last one
    - All new jobs run in nodes with new `Feature`, nodes reboot as old jobs end

Rollback



- Most important de-risking:
  - What happens if things go very wrong?
  - Roll back to previous image. Same mechanism.
    - Reboot FE nodes with old image, switch DNS name back
    - Redo the transition with the old BOS template as the input
    - Allow the whole system to roll back
  - *Low user impact*

Working Under Pressure

- *Nontrivial human considerations:*
  - Temptation to “just fix it” in the tight constraints of a normal DST
  - Heroic efforts are not sustainable
  - Crappy hack solutions are not desirable or sustainable

- *De-risking pressure-related errors and poor choices:*
  - Testing and scale testing take place over wider timeframes
  - Time to try and try again to get things right
  - Plenty of room to back out half-baked changes without jeopardizing others

Interruptions to Science

- It’s not just CPU time
  - Scientists are expensive.
  - Most of their expensive time on the system is getting things working
  - Taking the system away burns their interactive time, usually at the worst possible moment.
    - Trust us, it’s always just before a paper deadline
- Interruptions in configuration
  - Recompile needed? Can be a massive interruption
    - Get it done in pre-release testing time? Can happen in parallel

Summary

- Rollout/Rollback upgrades promise to keep users happy
- Systems are better maintained
  - Better testing
  - Better solutions to problems arising
  - Happier sysadmins
- Wasted time can be minimized
- System testbeds can be used for testing more invasive/exotic interventions with less interruption
- Rollbacks vastly reduce system downtime risk