

Systems Management Breakout Conclusions



Leaders:

**Bill Allcock
Alain Roy**

System management includes:



- Jobs
(SLURM, Cobalt, PBS, Torque, Condor...)
- Node health and testing
(INCA, RSV, NAGIOS, CACTI, Cerebro, Zenoss...)
- Change control
(CFengine, BCFG2, Puppet, RPM...)

Jobs



- Schedulers in use: Cobalt, Torque/MOAB, SLURM/MOAB, PBS Pro, LSF, Condor, SGE
- Best Practice: Use the same tools (like Torque/MOAB) across an entire site. Allow policies to differ between resource. Users and administrations benefit from the consistency.
- Best Practice: Provide well-defined, well-considered, openly published APIs to your software so that behavior can be modified at sites.

Jobs



- Challenge: It's important to get good performance in job launching. Open-source software mixed with proprietary hardware makes this harder, unless you have the right interfaces.
- New technology: How do we deal with a scientific workflow that needs to access computation, storage, network, etc?
- Challenge: Resiliency to failures gets more complex when we consider complete (long-running!) scientific workflows and increasing scale of systems.

Node Health and Monitoring



- Challenge: There is greater diversity in monitoring software than scheduler software.
- Challenge: It's hard to bring data together and correlate it across your systems
- Best practice: target your resources towards data management (federation), not data collection.
 - Separate data from data collection
 - We have (and will always have) wide variety of data collection systems, we need something common for data management/data formats.

Node Health and Monitoring (2)



- Challenge: A barrier to big improvements is that we have something that works for today and incremental improvements seem easier.
- Challenge: Knowing what to watch, getting the right information, bringing it together in an actionable way.
- Best practice: Maintain a historical database of failures, replacements, and maintenance, and periodically validate outages.

Change Management



- Tools in use: BCFG2, Cfengine, Puppet, OneSIS. (Fewer tools than monitoring: similar number of tools to scheduling. These are all open source tools, though some systems ship with proprietary tools.)
- Best practice: Makes changes in one known spot, then pushed out once it is right. Or a three-step process: test and development, qualification, then production.
- Best practice: Have a test and development system.
- Best practice: verify that changes happened, and monitor it occasionally.
- Best practice: Don't change during off-hours. Example: Tuesday-Thursday are good days, during business hours, not Friday-Monday.
- Best practice: have a well-defined process for deciding what changes should be made to the system, and when they should be made.
- Best practice: keep configuration in source-code repository (Subversion, etc) and treat it like software development. And keep it backed up.

Cross-Cutting



- **Communities**
 - (Best practice?) We need a community so we can agree on what our problems are and what we're trying to accomplish, so we can decrease the diversity of systems. If we speak with one voice, we have more clout in the vendor (or open source) community.
- **Support model**
 - Challenge: There is a trade-off between commercialization (get someone else to do it cheaper) and open-source (so we can do research and fix problems). Other models:
 - ✦ Task order: pay company to implement it
 - ✦ Convince vendor it will be useful to them in the future

Cross-Cutting (2)



- **Challenge: We need to manage diversity of our products.**
 - Sometimes there are too many overlapping tools, sometimes not enough diversity.
 - There is a tension between varied needs and shared experience and increased efficiency.
- **Suggestion: We should periodically review the state of the union of our products. We need a forum in which to do this.**
 - Some product areas (jobs) are more mature, and might focus on reducing diversity.
 - Some product areas (monitoring) are less mature, and may be more exploratory.