Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

NNSA
National Nuclear Security Administration

Design and implementation of a scalable monitoring system for Trinity

Cray User Group 2016
London, UK

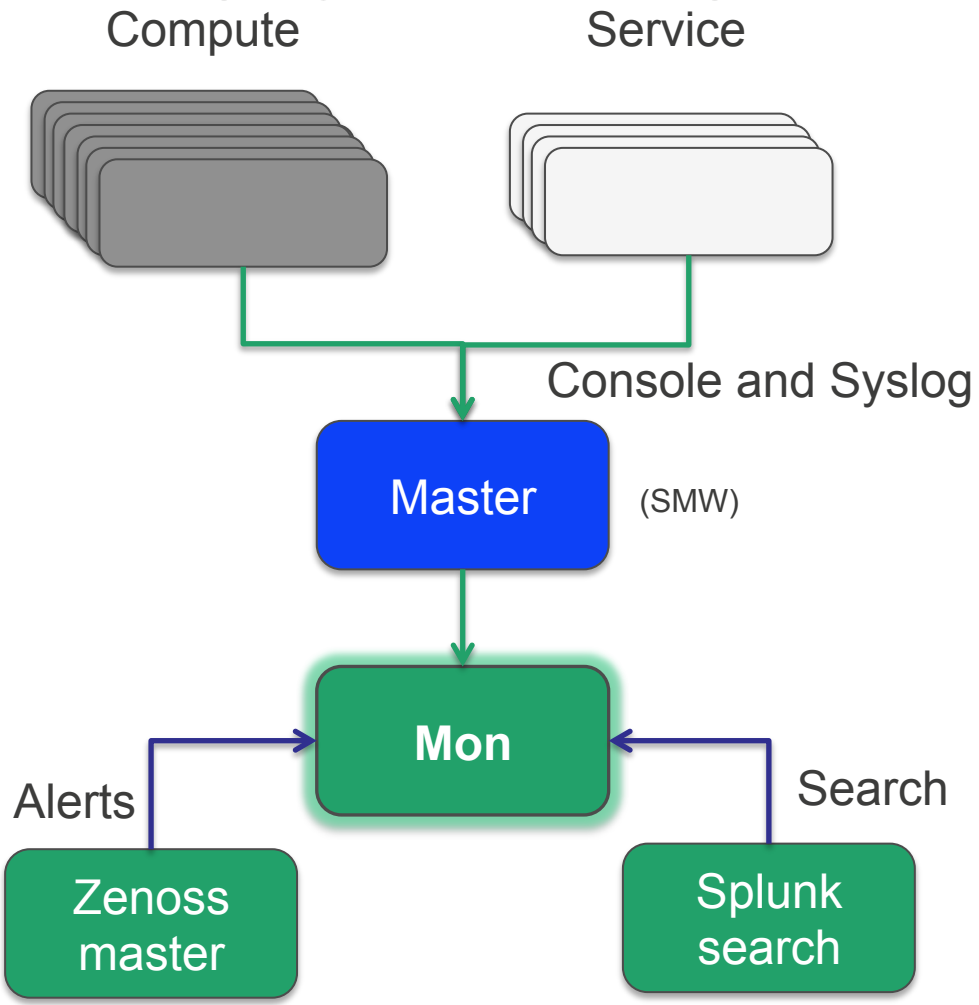
Adam DeConinck, A. Bonnie, K. Kelly, S. Sanchez, C. Martin, M. Mason - LANL
J. Brandt, A. Gentile, B. Allan, and A. Agelastos - SNL
M. Davis and M. Berry - Cray

12 May 2016

Agenda

- Review LANL's "legacy" monitoring stack, and why it won't work for *Trinity*
- Architecture of our new monitoring system
- Log analysis data flow
- In-band metric collection
- In-band monitoring overhead
- Early monitoring in production

“Legacy” monitoring stack



- **Simple, single-node monitoring architecture, standard across LANL clusters**
- **Cluster master (or SMW) aggregates console and syslog**
- **Forward to dedicated cluster monitoring server**
- **Mon server is a member of shared Zenoss and Splunk clusters**
  - Zenoss for alerts
  - Splunk for interactive search and debug

“Legacy” monitoring stack

- **Monitoring based mostly on log analysis**
- **Minimal “active” checks (temperature, IB fabric errors)**
- **Handles relatively low data volume**
  - 1600-node *Mustang* cluster, ~4.2 GB/month log data

- **This works well! For…**
  - Relatively small and static set of alerts
  - Detecting failed hardware
  - Alerting on well-understood software problems with simple log signatures
  - Doing basic searches and simple analysis on log data

Deficiencies in the legacy stack

- **Application performance monitoring**
  - Current stack is entirely system-oriented, little or no perf data
- **Complex analysis and visualization**
  - Neither Splunk or Zenoss is suitable for doing complex analysis or viz
  - Frequently we end up doing analysis on bulk logs anyway
- **No good integration with facilities monitoring**
  - Facility monitoring exists, but decoupled from platform monitoring
- **Flexibility and exporting data**
  - We keep finding new things to use monitoring data for! And don't know what we'll need to do in the future.
  - …but exporting from Zenoss/Splunk is difficult
  - ...and forwarding to additional destinations imposes more load

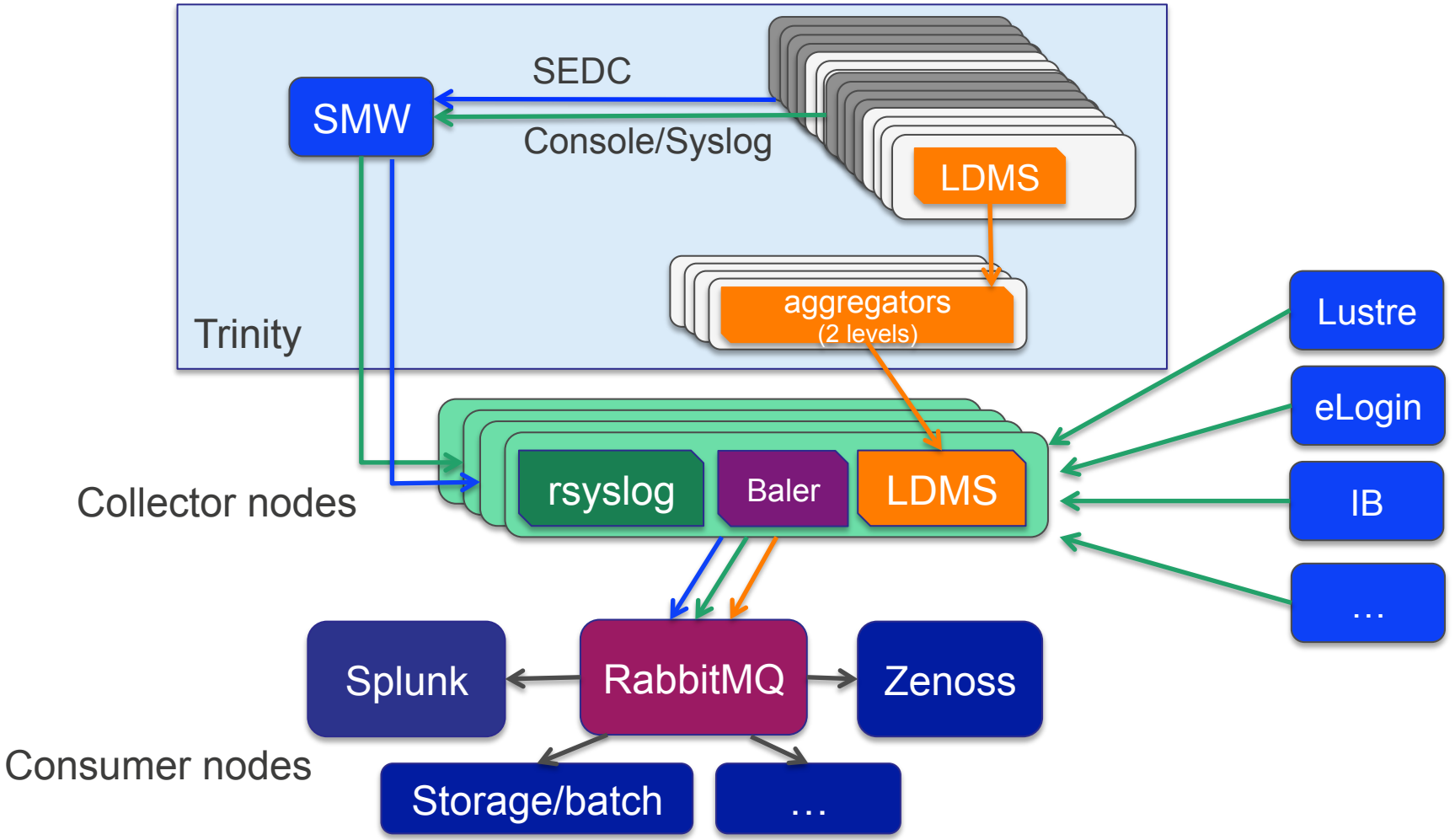
Trinity

- **Large Cray XC-40 deployed in two phases:**
  - 9,000+ Haswell compute nodes
  - 9,000+ KNL compute nodes
- **Very different from our previous deployments:**
  - New processor technology
  - Burst buffers (DataWarp)
  - New water cooling system
  - Power management
  - New software stack
    - SMW 8.0/CLE 6.0
    - KNL-related libraries, etc

Trinity is a major monitoring challenge!

- **New user-facing technologies: Haswells, KNLs, DataWarp**
  - Need better information on performance of the system for apps
- **More tightly coupled to the Facility**
  - Water cooling system – new to this facility
  - Power management extremely relevant – Trinity may use 8-10 MW
- **More data volume**
  - Much larger than any of our existing systems, planning to collect more data
- **More complex system, more difficult to debug**
  - SMW 8.0/CLE 6.0 is a major change in system management philosophy, both compared to past Crays and other clusters
  - Many more subsystems, data feeds, interactions
  - We don’t know what we don’t know… Need to be able to change our monitoring system easily as we go

Clustered monitoring system



Clustered monitoring system

- **Replace single monitoring node with a clustered solution**
  - Increased data volume and complexity
  - Redundancy of data path
- **Multiple node types all managed in the same cluster: data collectors, RabbitMQ nodes, data consumers**
- **Managed with the same tools we use to build our commodity HPC systems: Perceus, CFEngine, conman, powerman, …**
- **The rest of this talk walks through the various components of this system**

Clustered monitoring system

- **Console, syslog, and SEDC follow usual path through the SMW**
- **Additional metrics are collected on a per-node basis using LDMS**
  - LDMS data is transported off-system via aggregation nodes
- **All data from Trinity, plus attached systems like Lustre, eLogin, and the IB network, is collected by a cluster of collector nodes**
- **Data is forwarded from collector nodes to RabbitMQ message bus**
- **Consumer nodes subscribe to feeds from RabbitMQ to store and analyze data**
- **Estimate 4 TB/day total data, most of which is LDMS**
- **Collector and consumer nodes are managed using the same tools as our commodity clusters: Perceus, Cfengine, etc**

Agenda

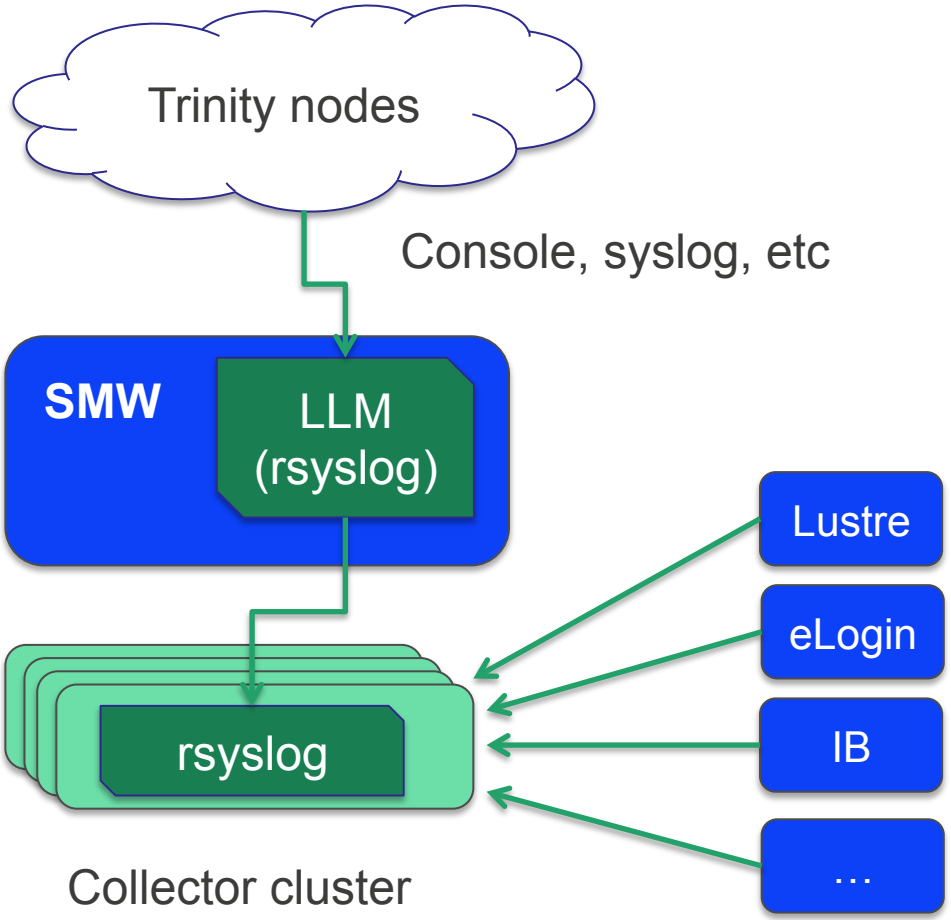
- Review LANL's "legacy" monitoring stack, and why it won't work for *Trinity*
- Architecture of our new monitoring system
- **Log analysis data flow**
- **In-band metric collection**
- **In-band monitoring overhead**
- **Early monitoring in production**

Log analysis data flow

Logs: system, console, application, …

- **Log analysis is still a key part of our monitoring**
  - Everything produces logs
  - Large existing library of filters, alerts, etc. based on past experience on *Cielo*
  - Syslog/LLM infrastructure provides a “one big feed” which is already transported from each node through to our monitoring system
  - Can easily inject arbitrary messages into logging feed
- **Where logs are written locally, we do our best to somehow import them into the syslog stream**
  - rsyslog *imfile* plugin

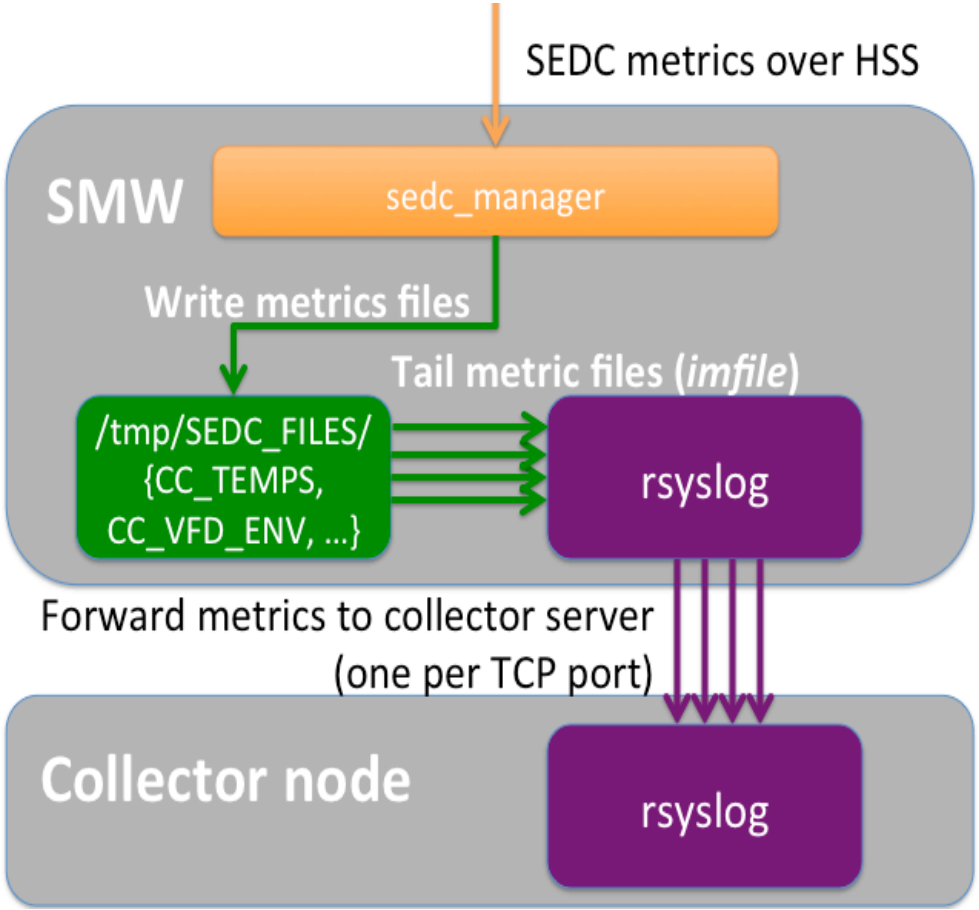
Log collection



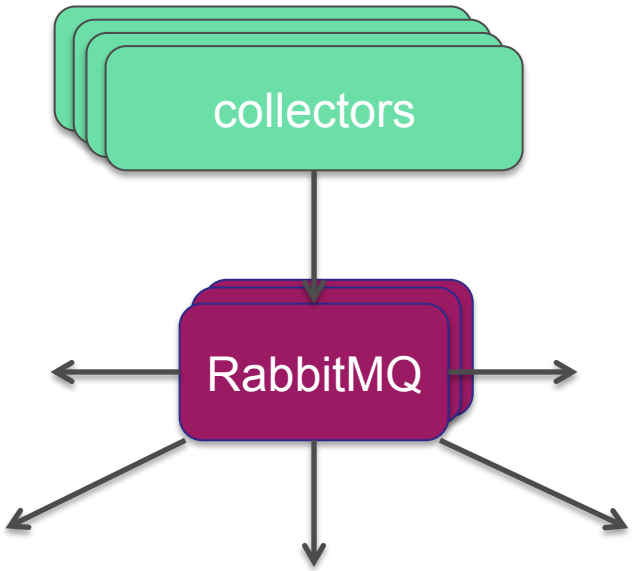
- **Logs are forwarded from Trinity through SMW to collector cluster**
  - SMW load is high! *rsyslog* frequently pegs at 100+% CPU usage
  - **TODO**: we want to move log forwarding to a different path which doesn't involve SMW
- **Multiple collector nodes for high load and redundancy**
- **Same cluster gets logs from all systems related to *Trinity***
  - Lustre, eLogin, IB fabric, etc

SEDC (a special case)

- **Really important for Facilities**
- **SEDC metrics also go through SMW logging path**
  - *sedc_manager* runs on the SMW
  - Write to flat files which can be re-ingested by *rsyslog*
- **One problem: SEDC lines have *no* metadata.**
  - To identify data, have to separate feeds or munge data
- **TODO: forward this data without involving the SMW**
  - UP01?

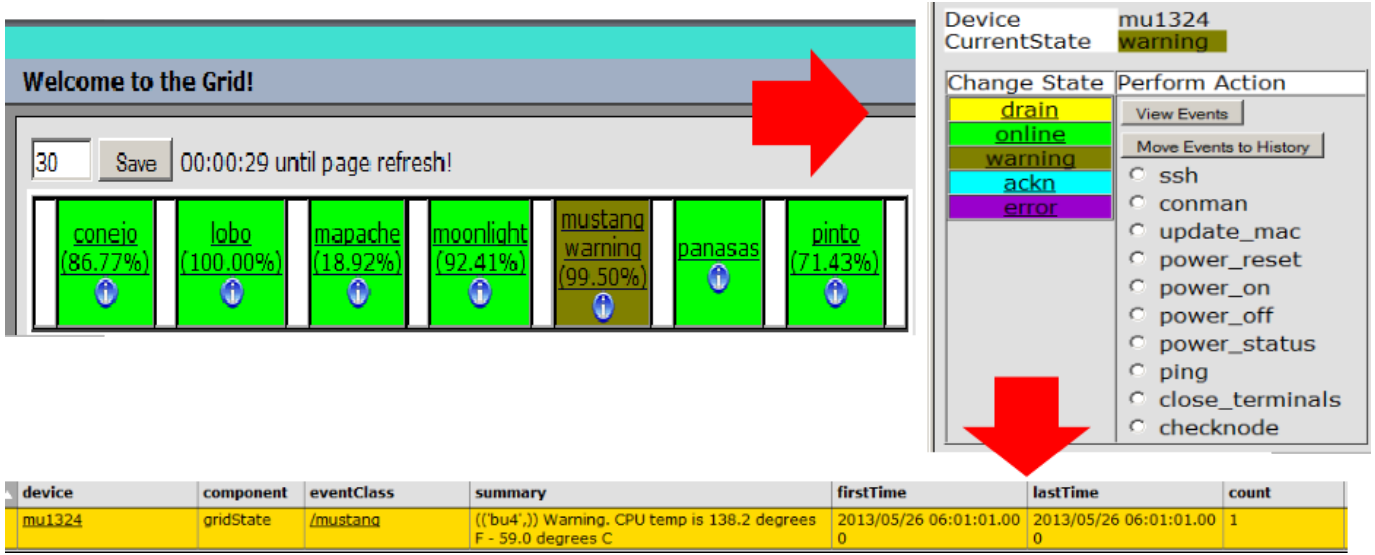


Distribution of log messages to collectors



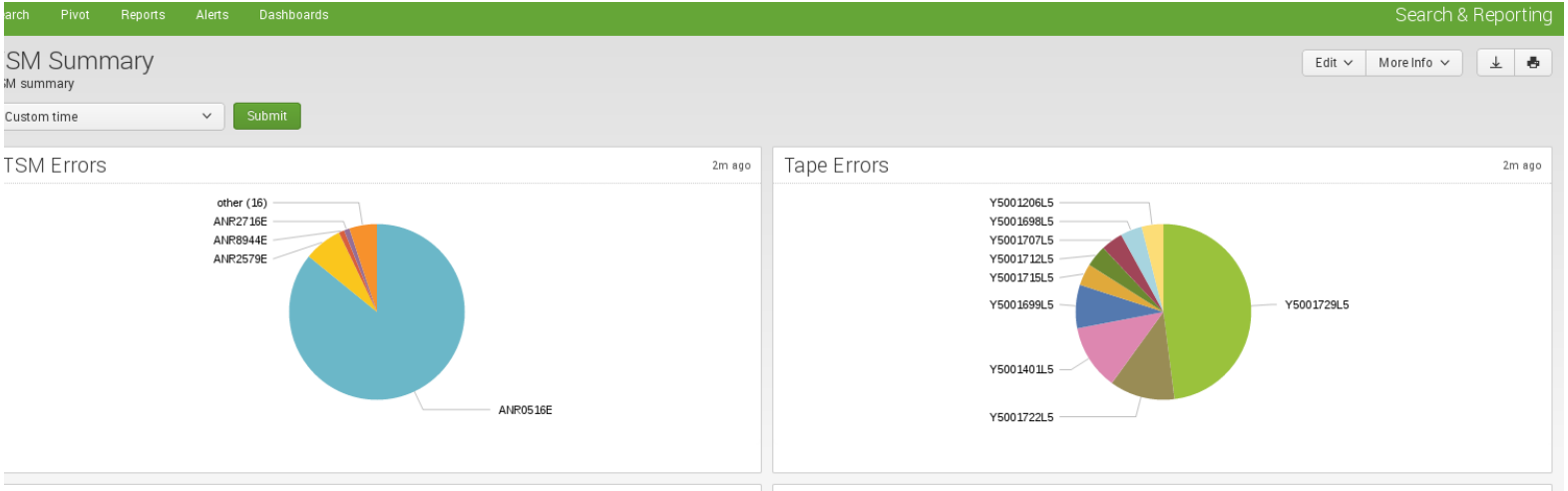
- **OK, we have a huge feed of log data. What do we do now?**
  - More importantly: what will we *want* to do in a year?
  - … we don't know. ☹
- **Many tools, many use-cases, same data**
- **Feed it all into RabbitMQ message bus**
  - Shared LANL infrastructure
- **Each analysis tool subscribes to the feeds it needs**

Operational alerting: Zenoss



- **Highly customized for LANL environment**
- ***Lots* of embedded institutional knowledge; many existing filters for basic issues will still translate to Trinity**
- **Generate real-time alerts, provide red/green dashboard for Ops staff**

Log searching and visualization: Splunk

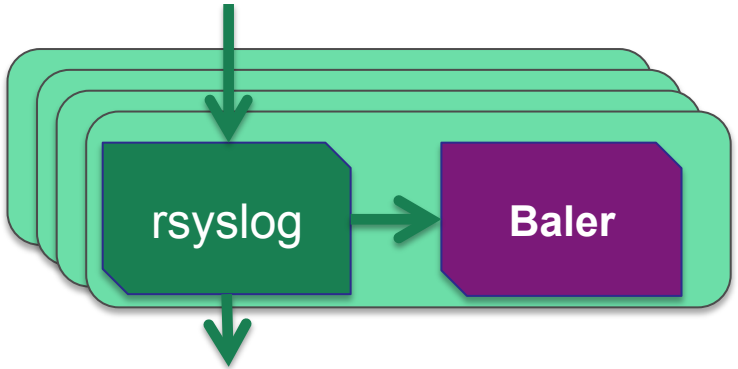


- **Splunk is essentially a search engine for time series log data.**
- **Provides a fast interface to search, visualize, and generate reports based on log data**
- **Used for developing topic-specific dashboards (i.e. filesystem, scheduler, or hardware error dashboards)**
- **Search engine used extensively to help debug problems on the system**

Pattern analysis and filtering: Baler

- **Special case in our architecture is *Baler*: automated log message pattern analysis**
- **Potentially reduce millions of log messages to thousands of *patterns* which can be filtered and searched**
  - Example: all sshd login messages look alike
- **New patterns potentially represent new issues or behavior**
- **Currently runs directly on collector nodes, not as a RabbitMQ consumer**

```
Example patterns:

280 * * - -  Node * interrupt *=*, *=*, *=* *[*]: * * *
    Processor Hot

283 * * - -  Node * power budget exceeded! Power=*,
    Limit=*, * Correction Time=*


Example messages corresponding to patterns 280
and 283 respectively:

bcsysd 2080 - -  Node 2 interrupt IREQ=0x20000,
    USRA=0x0, USRB=0x80 USRB[7]: C0_PROCHOT CPU 0
    Processor Hot

bcpmd 2140 - -  Node 2 power budget exceeded!
    Power=340, Limit=322, Max Correction Time=6
```



In-band metric collection

LDMS: Collection

- **Compute node resource metrics useful for understanding performance and detailed system behavior, but not included in syslog feed**
- **Exposed using on-node interfaces (e.g., /proc or /sys filesystems)**
- **Lightweight Distributed Metric Service (LDMS)**
- **Metrics collected include:**
  - CPU utilization
  - Memory utilization
  - Aries network counters (via Cray *gpcd* interface)
  - Power data
  - Lustre counters (e.g. opens, closes, reads, writes)
- **Can add LDMS plugins to collect other metrics**

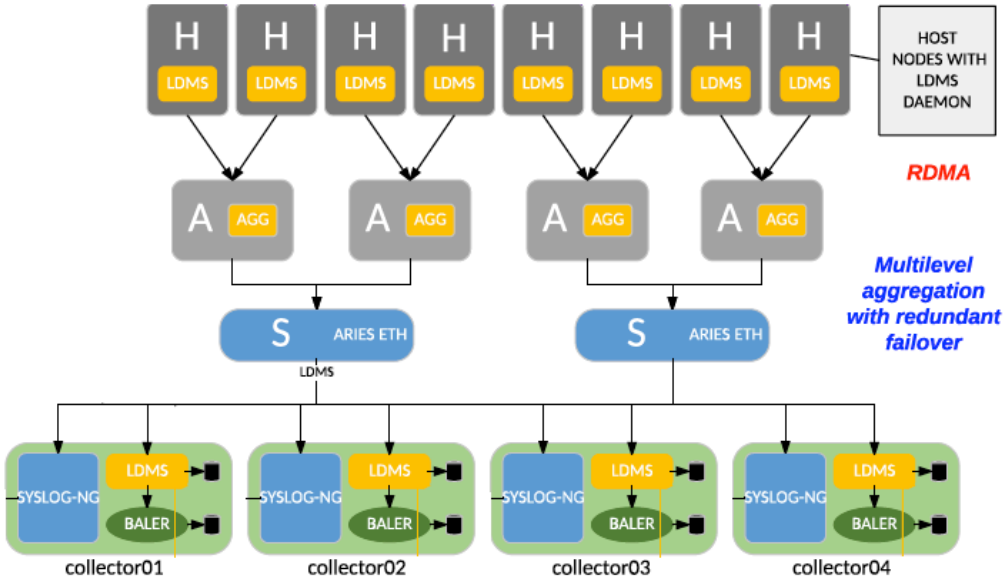
LDMS: Aggregation

- **Two modes for *ldmsd* daemon:**
  - Sampler (collecting metrics)
  - Aggregator (poll samplers for data)

- **Currently all metrics are being sampled at 1 Hz, except for power metrics which are sampled at 10 Hz**
- **Metrics are aggregated at 1 Hz**

- **Previous studies show maximum ratio ~16,000:1 samplers:aggregators on Cray Gemini metrics**
- **Trinity (full deployment) will include 19,000+ compute nodes, so we need at least 2 aggregators minimum; or 4 aggregators for redundancy**

LDMS: Multi-Level Aggregation



- **Problem: to transport LDMS data off system, we need service nodes with external interfaces. But Trinity has only two un-used service nodes!**
- **Three-level aggregation scheme:**
  - **L1:** 4x repurposed compute nodes poll samplers
  - **L2:** 2x service nodes poll L1
  - **L3:** Monitoring cluster polls L2 to extract data from cluster

LDMS data volumes

- **LDMS collection is currently too data-intensive to forward through the same RabbitMQ and log analysis pipeline as the rest of our logs**
  - In initial trials, we are collecting ~1.9 TB/day from Trinity Phase 1
  - Phase 2 will collect minimum of ~4 TB/day
  - More if we add more metrics!

- **Currently this data is stored directly on the monitoring cluster collector nodes, and periodically archived to external storage**
- **At some point, we plan to select some set of metrics to forward at a lower frequency through the rest of the pipeline**

In-band monitoring overhead

Impact of monitoring on user applications

- **While monitoring is useful for understanding system behavior, the system has actual work to do as well!**
- **Overhead of monitoring is potentially a major concern if it has an effect on user jobs**
- **On *Trinity*, thankfully, most data collection is out-of-band:**
  - Compute nodes do not run syslog
  - Console logs and SEDC collected over HSS, don't have on-node impact
  - Service nodes do log, but (except on login nodes) do not impact user code
- **LDMS data *is* collected in-band and could potentially impact user code**
- **Conducted a short study of several applications to determine impact**
- **Applications selected by Trinity Open Science users and program management**

HPCG

Condition	Run	Benchmark Time	All Reduce (AR) Min	AR Max	AR Avg
Baseline	1	603.81	19.63	79.12	74.65
Baseline	2	595.58	19.55	78.73	73.87
Baseline	3	603.59	20.64	75.25	70.85
w/Monitoring	1	598.55	22.17	78.01	72.86
w/Monitoring	2	594.84	19.35	77.24	72.84
w/Monitoring	3	596.51	20.16	75.84	71.07

- High performance conjugate gradient (HPCG) consists of operations such as sparse matrix-vector products.
- Expected to stress the memory subsystem and network communications.
- Three 10 minute runs of 9293 nodes, with 18568 processes with 16 threads each were run, both in baseline and with 1 second monitoring conditions.
- Benchmark average times with monitoring are actually 0.7% lower than the baseline average and the All Reduce times are comparable.

partisn

Condition	Cycle	Average Time	Min Time	Max Time
Baseline	1	50.53	50.41	50.66
Baseline	2	38.44	38.37	38.48
Baseline	3	38.26	38.22	38.32
Baseline	4	37.28	37.22	37.34
Baseline	5	37.33	37.30	37.36
w/Monitoring	1	50.42	50.29	50.49
w/Monitoring	2	38.47	38.45	38.49
w/Monitoring	3	38.29	38.26	38.31
w/Monitoring	4	37.28	37.26	37.32
w/Monitoring	5	37.36	37.34	37.38

- partisn is important code in the LANL workload. It is a deterministic neutron transport code .
- Seven runs of a partisn problem on 8192 nodes with 32 ranks per node were run: 3 runs were with monitoring at 1 sec and 4 runs were under baseline conditions. Each run had 5 cycles. Metric used was cycle times.
- Impact was 0.08% or less.

Nalu

Condition	Run	Walltime (min)
Baseline	1	8.34
Baseline	2	8.26
Baseline	3	8.40
w/Monitoring	1	8.37
w/Monitoring	2	8.33
w/Monitoring	3	8.35

- Nalu is a signficant code in the SNL workload.
- It is an adaptive mesh, variable-density, acoustically incompressible, unstructured fluid dynamics code. Should be sensitive to both node and network slowdown.
- For this test, we ran 3 concurrent instances of a 65K core simulation that was being run during the Open Science period on Trinity. Due to time limitations, however, we ran a limited set of timesteps starting from a restart file. Metric used was the baseline time.
- The average run time with monitoring had a 0.3% increase, however all monitoring run times were with the the minimum and maximum run times of the baseline cases.

psnap

Condition	Run	Per node avg slowdown	min slowdown	max slowdown
Baseline no barrier	1	0.217 +/- 0.012	0.200	0.243
Baseline no barrier	2	0.217 +/- 0.011	0.199	0.245
Baseline no barrier	3	0.217 +/- 0.011	0.199	0.247
w/Monitoring no barrier	1	0.237 +/- 0.009	0.209	0.253
w/Monitoring no barrier	2	0.237 +/- 0.009	0.205	0.254
w/Monitoring no barrier	3	0.238 +/- 0.009	0.209	0.253
Baseline w/barrier	1	0.226 +/- 0.012	0.204	0.249
Baseline w/barrier	2	0.226 +/- 0.012	0.202	0.251
Baseline w/barrier	3	0.223 +/- 0.011	0.204	0.248
w/Monitoring w/barrier	1	0.238 +/- 0.011	0.207	0.259
w/Monitoring w/barrier	2	0.238 +/- 0.011	0.212	0.259
w/Monitoring w/barrier	3	0.231 +/- 0.011	0.209	0.271

- **PSNAP is used to measure OS jitter.**
- **3 runs were performed under baseline and monitoring conditions. We ran 100000 loops of 1000 microsecs with and without a barrier every 100 loops. The runs were performed on 9216 cores, 32 cores per node. The metric of comparison is the change in slowdown, where slowdown is the actual loop times as compared to the ideal loop time.**
- **With 1 sec monitoring, the slowdown increased by < 0.02% above the baseline slowdown, which we deemed acceptable.**

Development of monitoring in production

Running in production

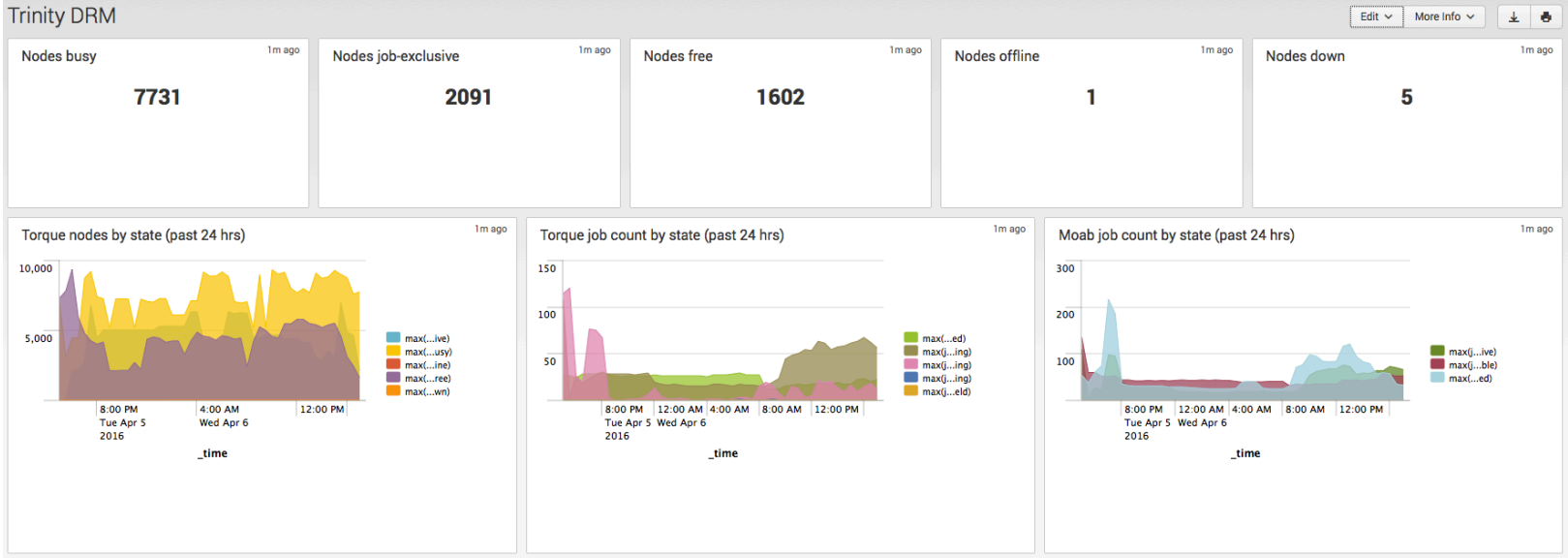
- *Trinity* is still in early stages
- We are still learning a *lot* about the system as it is deployed
- Making changes as we go, both in generating data and in
- This section covers a few things we've been working on during the early stages of running *Trinity*

Service health checks

- Compute nodes run NHC before and after jobs, but service nodes run no periodic checks be default
- We need active checks of system services to ensure they are still operating as expected!
- Biggest offenders: Lustre, Moab, DataWarp
- Health check scripts run in *cron*, drop pass/fail messages per-test into syslog
- Notifications generated using regular Zenoss stream

```
PASS: kernel version matches expected value
PASS: hostname matches expected value
PASS: autofs active and enabled
PASS: rsyslog active and enabled
PASS: sshd active and enabled
PASS: df -h works on /lustre/scratch5
PASS: lfs df returns within five seconds
PASS: local filesystems are mounted
PASS: lustre mounted
PASS: ls -l works on /users/testuser
PASS: ls -l works on /usr/projects/testproject
PASS: pbsnodes returns
FAIL: pbsnodes has 9337 nodes in trinity
FAIL: mdiag shows tr-drm as the moab server
FAIL: showq –blocking returns
FAIL: showq returns
FAIL: showres returns
FAIL: showstate returns
12 TESTS SUCCEEDED and 6 TESTS FAILED
```

Subsystem-specific visualizations



- **Zenoss GRID dashboard is a “whole system” view, red/green lights**
- **Early stages of developing dashboards for specific Trinity subsystems**
  - Boot, Scalable Services, Scheduling, DataWarp, ...
- **Currently using Splunk, trying to determine best solutions**

Data-driven operations

- **Improve run-time integration of multiple data sources to improve operations**
  - I.e., dashboards based on known log messages, LDMS and SEDC metrics, and showing new patterns found by Baler
- **Correlate numerical and text-based data**
- **Make LDMS data available to users on a per-job basis**
- **Understand network congestion to improve performance**

Conclusions

- The scale and complexity of Trinity has motivated a re-design of our monitoring system
- Monitoring cluster built with the same tools as our commodity HPC systems for scalability and flexibility
- Integration of multiple data sources in the monitoring cluster, and distribution using RabbitMQ message broker
- Flexible collection of data consumers for monitoring including Zenoss, Splunk, and Baler
- Continuing work on health checks, visualization, and data-driven operations

What's next?

Moving *Trinity* into full production is still ongoing. Monitoring is expected to evolve accordingly.

- **Eliminate SMW from data transport path wherever possible**
- **Improve redundancy and high availability for handling log data**
- **Analysis of Trinity logs with Baler to characterize "normal" system events**
- **Choose/develop additional analysis and visualization tools and attach to RabbitMQ broker**

Acknowledgements

- Almost all of LANL HPC has been involved in Trinity deployment, which has been closely entwined with the deployment of Trinity monitoring
- LANL production support team, including Vaughn Clinton, Susan Coulter, Gregory Geller, Daryl Grunau, David Morton, Paul Peltz, Conor Robinson, Quellyn Snead, Graham Van Huele, and Jim Williams
- Cray site personnel, including Richard Dimock, Greg Hamilton, Bruce Ensberg, Jack Tapie, Kevin Stroup
- Developers of LDMS at SNL and Open Grid Computing

Questions?

ajdecon@lanl.gov