The NERSC Data Collect Hotel

Thomas Davis
Cary Whitney

2/28/17
The mission of the National Energy Research Scientific Computing Center (NERSC) is to accelerate scientific discovery at the DOE Office of Science through high performance computing and data analysis.

NERSC is the principal provider of high performance computing services to Office of Science programs — Magnetic Fusion Energy, High Energy Physics, Nuclear Physics, Basic Energy Sciences, Biological and Environmental Research, and Advanced Scientific Computing Research.

Computing is a tool as vital as experimentation and theory in solving the scientific challenges of the twenty-first century. Fundamental to the mission of NERSC is enabling computational science of scale, in which large, interdisciplinary teams of scientists attack fundamental problems in science and engineering that require massive calculations and have broad scientific and economic impacts. Examples of these problems include photosynthesis modeling, global climate modeling, combustion modeling, magnetic fusion, astrophysics, computational biology, and many more.
Shyh Wang Hall, is a 149,000 square foot facility built on a hillside overlooking the UC Berkeley campus and San Francisco Bay. This building houses one of the most energy-efficient computing centers anywhere, tapping into the region’s mild climate to cool the supercomputers at the National Energy Research Scientific Computing Center (NERSC) and eliminating the need for mechanical cooling.
Our machine room is unique.

- Shyh Wang Hall sites about 200 yards from the Hayward Fault Line.
  - Machine room floor consists of two very large tables with moats.
- We have no chillers.
  - We have tower water
    - Water is provided to the floor through plate-style heat exchangers
    - We depend on the SF Bay area temperate climate.
- Our power comes from Western Area Power Administration (WAPA), not PGE.
  - We can cause large scale power swings
    - Currently 2-4MW in range
    - N9 system could be in the 10-15MW range.
- Long periods of downtime also cause problems.
  - Anything more than 24hrs can cause problems.
Ivy Bridge
30
134,064 Cores
357 TB Memory
5586 Nodes

Cray Dragonfly topology 23.7 TB/s bisectional bandwidth

For only 2 MW of peak power

Lustre Scratch disk space
7.56PB

168 GB/s
Cori, a Cray XC40 based system

<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th>KNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>16,128</td>
<td>632,672</td>
</tr>
<tr>
<td>Memory</td>
<td>203 TB</td>
<td>1 PB</td>
</tr>
<tr>
<td>Nodes</td>
<td>2004</td>
<td>9304</td>
</tr>
</tbody>
</table>

Burst Buffers: 1.8PB SSD
Dynamic storage: 700 GB/s

Cray Dragonfly topology 45 TB/s bisectional bandwidth

For only 7 MW of peak power

Lustre Scratch disk space: 30PB

700 GB/s
HPSS archive system

- **Data stored in archive system:** 90 PB, >179 million files
- **Growth Rate:** 1 PB/month
- **Current Maximum capacity:** 240 Petabytes.
- **Buffer (disk) cache:** 288 Terabytes.
- **Average transfer rate:** 100 MB/sec
- **Peak measured transfer rate:** 1 GB/sec
Compute Floor
So, where does all the data come from? And Why?
Data Sources

Data Collect
ELK Stack
Multiple Logstash

Elastic cluster

API

Kibana graphical interface
for data exploration and display

Closed shards that are archived
and can be loaded later by other
elastic cluster instances.
This included Mac and PC
stacks.

RabbitMQ cluster

ElasticAlerts

Grafana

Redis DB for Lustre and other data
for immediate access

Redis DB for power and cooling
tombstone database

Power and environmental
data. Direct from sensors
or a collector. Modbus and
one wire, text and json
format

Per system forwarding
logstash

collectd

Metricbeat

Filebeat follower for
text files generated
by other programs

custom customer
methods (text or json)

LDMS
Prometheus
inputs

 Lustre Filesystem data
GPFS Filesystem data
SLURM job data

Other output methods can be added
such as mysql, mongoDB, etc.
This can be a subset of the data collected
or the full data.
This data can also be shipped to other
centers either on ssl or not.

Archive syslog data to disk
Possible custom methods

Center Syslogs

Weather Station

SEDC Cray
Power and
Cooling

Building
Automation
System

Networking IP and IB
not gathered from
nodes

Filebeat collector
<table>
<thead>
<tr>
<th></th>
<th>Size (GB)</th>
<th>doc count (M)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modbus</td>
<td>15.4</td>
<td>99.7</td>
<td>Serial based industrial devices 2500 PDU stripes and 849 PDU panels and substation</td>
</tr>
<tr>
<td>collectd</td>
<td>108.75</td>
<td>807.8</td>
<td>Linux system stats</td>
</tr>
<tr>
<td>SEDC</td>
<td>27.6</td>
<td>261.4</td>
<td>Cray power, environmental and job</td>
</tr>
<tr>
<td>Syslog</td>
<td>4.25</td>
<td>21.95</td>
<td>Logs from all systems/devices of the center</td>
</tr>
<tr>
<td>weather</td>
<td>0.017</td>
<td>0.044</td>
<td>Davis Weather station outside</td>
</tr>
<tr>
<td>onewire</td>
<td>0.940</td>
<td>5</td>
<td>Computer room temperature network over 1800 sensors</td>
</tr>
<tr>
<td>upmu</td>
<td>0.46</td>
<td>0.164</td>
<td>High resolution power monitoring</td>
</tr>
<tr>
<td>ION</td>
<td>0.206</td>
<td>1.9</td>
<td>Building substation power monitoring</td>
</tr>
<tr>
<td>Total</td>
<td>160</td>
<td>1.2B</td>
<td></td>
</tr>
</tbody>
</table>
We even have a seismophone
Why

- **Power**
  - Used for capacity planning
  - Also useful to diagnose problems.
- **Environmentals**
  - Air Temperature
  - Water Temperature
  - Water Pressure
  - Water Flow Rate
- **Performance monitoring**
  - Disk I/O
  - Network I/O
  - Memory usage
  - CPU Usage
- **Security**
- **Future Exascale and beyond planning**
  - All of the above is used to plan, procure, build or remodel for the next generation systems.
How we use the data.
How we use this data
Cori’s Cooling performance
B59 Power
Meter Displays
Performance Metrics
Threat Analysis
Talk, talk, talk..
What we are going to talk about

- Our data collect system
- Long term archiving of data.
- Using hot, warm, and cold storage.
- Configuring elasticsearch to support this model.
- Snapshot and restore.
Long term Archiving

NERSC
Warning! Danger!

Everything we show you today is for Elasticstack v5. Many of the concepts are the same for previous versions of the Elasticstack, but some of the terms have changed between major versions.

We make no guarantee that any of this will work for you.

You must do tests of any system to ensure proper operation.
What does ‘long term’ mean

- Months, Years or even decades.
  - Must be readable forever.
  - Can be retrieved and restored at any time.
- Useful for long term modeling
  - System modeling
  - Machine room modeling
  - Mechanical models
- Helps to answer ‘What if’ questions
How we achieve this

- **We use a hot, warm, and cold architecture**
  - Hot nodes are our ingest and short term storage nodes
    - Days, even weeks of data
  - Warm nodes are our medium/archival storage nodes
    - Weeks, months, up to a year of data
  - Cold storage is done using a combination of technologies
    - HPSS (High Performance Storage System)
      - A very large tape storage system, with multiple very fast (10G+ jumbo frame) connections.
        - This is also a storage system used by everyone at NERSC.
    - Elasticsearch snapshot/restore
    - A large GlusterFS based filesystem
    - Elasticsearch Curator
  - Elasticsearch node attributes
The databases we use.

We use elasticsearch as our long term database.
- Time series metric type data
- System logs
- Events/Annotations

Redis is used as for several other functions.
- Tombstone database, AKA “last known value”
- Configuration
- Python RQ (Task queuing for the collectors)
- Time series caching

MariaDB is used for several support programs.
- Grafana, Opendcim, etc.

Postgresql
- BMS
A Hot storage node has a drive subsystem configured for speed instead of capacity.

- **IE, SSD/NVME based drive systems.**
  - We use 1TB sata drives at this time.
  - PCI based drives are available if desired.

A Warm storage node has a drive subsystem configured for capacity instead of speed.

- **IE, a RAID5 array of cheap, large capacity drives**
  - In our case, these are 5ea, 2.5” 2TB drives Linux software raid5 array.
  - combined with a SATA SSD drive
  - Both sets of drives are combined into one large drive using lvm-cache.

Cold storage is where the data takes time measured in seconds, minutes or even hours to access.

- This is our large glusterfs based global filesystem
- We also copy data to/from a HPSS archive system for long term storage.
The System
Data Collect Cluster

- **8 ea Supermicro Fat Twin 4u chassis**
  - 8 nodes per chassis
    - Minimum of 64GB per node.
    - 16 CPU cores
    - 10GB interface into a 10GB switch
    - Some nodes have just 1TB of SSD drive space.
    - Some nodes have also 5ea, 2.5” 2TB drives.

- **Software**
  - Centos 7 based.
  - Not all nodes are used for Elasticsearch.
  - Ovirt 4.1 is run to provide a VM service.
  - Rancher combined with VM’s from Ovirt is used to run the data collect.
  - Several elasticsearch nodes (client and master) are run as VM’s.
    - 3 master nodes
    - 3 client nodes pooled using Consul
    - Kibana, Grafana client nodes using client node pool.
      - No elasticsearch client runs on these nodes.
  - 19 ea Hot storage nodes
  - 10 ea Warm storage nodes
Supermicro FatTwin
Elasticsearch configuration
Now that we have defined our hot/warm storage nodes, this is where node attributes in Elasticsearch comes in.

We define two types of attributes

- An attribute that defines what type of node
  - This can be either ‘ssd’ or ‘archive’.
- An attribute that defines physical location data of the node.
  - Normally based on a chassis, ie ‘c0’.

The attributes are used by the system to place the indexes on the correct node. We do not want ingest data going to an archive node, and we do not want to use a SSD node for archival data.
node:
  data: true
  master: false
  name: ${HOSTNAME}
  attr:
    chassis_id: c0
    tag: ssd
path:
  data: /ssd/elastic
  repo: /glusterfs/ec0/es5
node:
  data: true
  master: false
  name: ${HOSTNAME}
attr:
  chassis_id: c0
  tag: archive
path:
  data: /data/elastic
  repo: /glusterfs/ec0/es5
Cold Storage

- Several technologies
  - HPSS.
    - Uncompressed data is best
    - We let the tape drives do the compression
  - Dual 10g Jumbo framed interfaces into this system
  - Capable of over 5GB/s transfer rates
  - GlusterFS
    - Elasticsearch snapshot/restore needs a global filesystem.
  - Elasticsearch snapshot/restore
  - Elasticsearch Curator v4
  - Shell scripts
  - Rundeck to run jobs
    - Cron can also do this.
Scripts
Curator config set an archive tag

actions:
  1:
    action: allocation
    options:
      key: tag
      value: archive
      allocation_type: require
      wait_for_completion: False
      timeout_override:
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: regex
        value: .*
        exclude:
      - filtertype: age
        source: creation_date
        direction: older
        unit: days
        unit_count: 4
        exclude:
Snapshots

• One Snapshot, One Repository per daily index.
  • We create repo’s on a per-index basis
  • Never shared.
  • One per day, one per index.
  • Each index becomes a tar file
    • Not compressed due to how the tape storage unit works.
  • Built using
    • Curator 4.0
    • Bash scripts
    • Large glusterfs volume
      • 30TB usable space
      • Built using erasure codes, not replication.
      • Glusterfs file sharding for performance.
Daily cold storage routine.

1. Change tag to archive on select indexes over 48hrs old
2. Forcemerger all indexes from the past 2 days
3. Get list of indexes created in the past 24 hours.
4. Create snapshot repository for index
5. Create snapshot
6. Wait for cluster to catch up
7. All indexes completed?
   - Yes: Create tar archive of snapshots
   - No: Go back to step 3
8. Store tar archive in HPSS
9. Sleep!
#!/bin/sh
MASTER="es5-client-pool.service.consul"
curator --config /home/tdavis/.curator/curator.yml /home/tdavis/curator/archive-allocation
curator --config /home/tdavis/.curator/curator.yml /home/tdavis/curator/force-merge
INDEXS=$(curator_cli --host $MASTER show_indices --filter_list '[["filtertype":"age","source":"creation_date","direction":"older","unit":"days","unit_count":2],
  ["filtertype":"age","source":"creation_date","direction":"younger","unit":"days","unit_count":3 ]]')
grep -v ".monitoring" 
for INDEX in $INDEXS
do
echo index: $INDEX
LOCATION="/glusterfs/ec0/es5/"$INDEX
curl --silent -XPOST http://$MASTER:9200/_snapshot/$INDEX
  -d '{ "type": "fs", "settings": { "location": "/glusterfs/ec0/es5/"$INDEX'", "compress": "true", "max_snapshot_bytes_per_sec": "200m" } }'
echo
curl --silent -XPOST http://$MASTER:9200/_snapshot/$INDEX/$INDEX
  -d '{ "indices": "$INDEX", "ignore_unavailable": "true", 
  "include_global_state":false }'
echo
sleep 300
done
YM=$(date "+%Y.%m")
DIR=/glusterfs/ec0/es5/archive
if [ ! -d /glusterfs/ec0/es5/archive/$YM ];then
  mkdir -p $DIR/$YM
fi
for INDEX in $INDEXS
do
echo Creating tar file $INDEX.tar
cd /glusterfs/ec0/es5/archive
  tar cf $DIR/$YM/$INDEX.tar $INDEX
done
ssh d8-r13-c4-n8 /root/xfer.sh
for I in $INDEXS
do
curl --silent -XPOST http://$M/_snapshot/$I \
    -d '{ "type": "fs", "settings": { "location": "/glusterfs/ec0/es5/'$I'",
    "compress":"true","max_snapshot_bytes_per_sec":"200m" } }'
    echo
curl --silent -XPOST \
    http://$M/_snapshot/${I}/${I}?wait_for_completion=true \
    -d '{ "indices": "'$I'", "ignore_unavailable": "true",
    "include_global_state":false }'
    echo
    sleep 300
done
Restoring indexes

1. Get index snapshot(s) from HPSS
2. Re-create index snapshot directory
3. Restore index snapshot to snapshot directory
4. Start restore of index snapshot
5. Wait for index snapshot to finish restore

- All indexes restored?
  - Yes
    - Profit!
#!/bin/sh
MASTER="es5-client-pool.service.consul:9200"

INDEXES=$( echo *.tar )

for IDX in $INDEXES
do
  INDEX=$(echo $IDX | awk -F '{ print $1 "." $2 "." $3 }')
  LOCATION="/glusterfs/ec0/snap/"$INDEX
  TAR="/glusterfs/ec0/elasticsearch/restore/"$INDEX".tar"
  curl -XPOST http://$MASTER/_snapshot/$INDEX \
      -d '{ "type": "fs", "settings": { "location": "/glusterfs/ec0/snap/"$INDEX"", "compress": true } }' \
  echo "restoring tar file into snapshot.."
  tar xf $TAR
  curl -XPOST http://$MASTER/_snapshot/${INDEX}/${INDEX}/_restore \
      -d '{ "indices": '"$INDEX"", "ignore_unavailable": "true", "include_global_state": false }' \
  echo "sleeping for 10 seconds.."
  sleep 10
done
while (true)
do

STATUS=$(curl -s -X GET http://$MASTER/_cluster/health?pretty=true | \
  grep "status" | awk '{ print $3 }' | cut -f1 -d",")
RELOCATING=$(curl -s -X GET http://$MASTER/_cluster/health?pretty=true | \
  grep "relocating_shards" | awk '{ print $3 }' | cut -f1 -d",")
INITIALIZING=$(curl -s -X GET http://$MASTER/_cluster/health?pretty=true | \
  grep "initializing_shards" | awk '{ print $3 }' | cut -f1 -d",")
UNASSIGNED=$(curl -s -X GET http://$MASTER/_cluster/health?pretty=true | \
  grep "unassigned_shards" | grep -v "delayed" | awk '{ print $3 }' | cut -f1 -d",")

echo STATUS: $STATUS RELOCATING: $RELOCATING INITIALIZING: $INITIALIZING UNASSIGNED: $UNASSIGNED

if [ $STATUS == '"green"' -a $RELOCATING == 0 -a $INITIALIZING == 0 -a $UNASSIGNED == 0 ]
  then
    break
fi

sleep 2
done
Sunset at Shyh Wang Hall
Questions