Managing a Heterogeneous Cluster

Kyle Hutson
Kansas State University
Manhattan, Kansas
kylehutson@ksu.edu

Dan Andresen
Kansas State University
Manhattan, Kansas
dan@ksu.edu

Adam Tygart
Kansas State University
Manhattan, Kansas
mozes@ksu.edu

Dave Turner
Kansas State University
Manhattan, Kansas
daveturner@ksu.edu

ABSTRACT

Most HPC clusters are purchased with a large quantity of identical hardware, which is maintained through its lifecycle and then another HPC cluster takes its place. However, some clusters, like ours, are maintained by frequently adding new hardware, which is then integrated into the system. Over the years, the cluster has grown to include 300+ compute nodes with 8000+ cores from 6 vendors, spanning 5 generations of CPUs; 7 network technologies from 6 switch vendors (1Gbps-100Gbps, including Ethernet, Infiniband, and OmniPath); 102 GPUs (3 different GPU models); 28 storage nodes (3+ PB raw storage); and 7 virtualization nodes hosting 65 VMs.

Having such a diverse system has significant advantages, although the management is more difficult. This paper outlines our strategy of managing this very heterogeneous and complex system. Topics covered include software optimization, consistency of operating system updates, identity management, resource prioritization, network infrastructure, storage, and management of non-compute-intensive resources. Our combination of open source and internally developed software used to manage this cluster are a model to other heterogeneous systems and to smaller clusters which have not expanded because of management worries.

CCS CONCEPTS

• **Social and professional topics → Management of computing and information systems**;

KEYWORDS

system administration, heterogeneous clusters, resource management

ACM Reference Format:
Kyle Hutson, Dan Andresen, Adam Tygart, and Dave Turner. 2019. Managing a Heterogeneous Cluster. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3332186.3332251

1 INTRODUCTION

Managing resources on a large, homogeneous HPC cluster is common, and mostly a solved problem. However, managing resources on a smaller, heterogeneous, resource-constrained HPC cluster is more challenging and largely done on an ad hoc basis. Through many years of managing a heterogeneous cluster, we have developed several processes, which can be applied to other similar systems.

Beocat is the primary research HPC cluster at Kansas State University. It is run by the Institute for Computational Research in Engineering and Science, which is a function of the Computer Science department. Beocat is available to any educational researcher in the state of Kansas (and his or her collaborators) without cost. Priority access is given to those researchers who have contributed resources.

Many clusters, when they have grown to the size of Beocat move to a more traditional, homogeneous model. As of the time of writing, Beocat includes 343 compute nodes, 28 storage nodes, 8368 compute cores, 3.15 PB of raw storage, 65 virtual machines, 7 network fabrics, 34 GPU-enabled nodes with between one and four GPUs each (102 total GPUs), and 7 virtualization hosts for 1777 user accounts. We, consciously, have not moved to a homogeneous model for a variety of reasons. (a) There is no central funding source, which can promise stability for the future. Most funding comes from small grants to working groups and startup packages. In return, the source of that funding need to be guaranteed immediate access to the resources they subsidized. (b) By purchasing more compute nodes more than once a year, we can purchase some of the fastest available systems, rivaling some of the world's largest supercomputers on a per-node basis. (c) Older nodes are often still useful well past their three-year warranty period, and are often used for both student projects and high-throughput codes. Beocat is maintained by only three full-time employees (two system administrators and one application scientist) in addition to the part-time director. Our processes have enabled us to be very efficient in the use of the resources we have.

Most processes are built on existing open-source solutions, but we put them together in ways that allow us to manage Beocat quickly and easily. We believe we can serve as a model to other institutions, which are expanding heterogeneously.

PEARC '19, July 28-August 1, 2019, Chicago, IL, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3332251

2 RELATED WORK

Managing Beocat could not happen without many open-source software packages, which we use and have adapted for our own use. It is the integration and effective use of these many programs, which enable us to manage Beocat efficiently. The following is a list of the most notable projects we leverage, many of which will likely be very familiar to this paper’s audience.

- Ansible (https://www.ansible.com/)
- Ceph (https://ceph.com/)
- EasyBuild (https://github.com/easybuilders/easybuild)
- Ganeti (http://www.ganeti.org/)
- Ganglia (http://ganglia.info/)
- GitLab (https://about.gitlab.com/)
- Icinga (https://icinga.com/)
- iPXE (https://ipxe.org/)
- Lmod (https://sourceforge.net/projects/lmod/files/)
- MediaWiki (https://www.mediawiki.org/)
- openDCIM (https://www.opendcim.org/)
- Request Tracker (https://bestpractical.com/request-tracker)
- Slurm (https://slurm.schedmd.com/)

There are no well-known resources available to those administering growing and heterogeneous clusters.

3 OUR APPROACH

When maintaining a heterogeneous resource, we keep as much consistency as possible with the disparate systems. When that is not possible, we automate as much as possible. Several examples and methodologies are detailed below.

3.1 Naming Conventions

Our naming convention consists of the “class” of the compute node, roughly equivalent to the CPU architecture, followed by a number. Simply knowing that a “mage” is a large-memory westmere node, but a “wizard” is a skylake node gives the system administrator much insight when a failure occurs, as to whether the system is new or old, in warranty or not, and which vendor to contact for support. Similarly, we allocate IP addresses based on these node classes, in a manner that makes more sense to the administrators than for efficiency on a technical level. As an example and using the classes mentioned earlier, although all of our compute nodes occupy the same IP space (10.5.0.0/16), we use the 10.5.14.xxx space for mages and 10.5.18.xxx space for wizards, so mage01 has the IP address of 10.5.14.1 and wizard25 has the IP address of 10.5.18.25. Our management network, used for out-of-band management (Dell’s iDRAC, Lenovo’s IMM2, any IPMI, etc.) uses the 10.8.0.0/16 network, with the last two octets being identical to its compute-network IP. The result is that if we have the knowledge that wizard25 has an IP of 10.5.18.25, we also know that wizard23 has an IP of 10.5.18.23, and its management IP is 10.8.18.23.

3.2 Slurm Configuration Mangement

We use slurm as our job scheduler. When a system administrator for a homogeneous cluster generates a slurm configuration file, they likely write it with a shell script, which gives the same line with only a minor change for each node. In our heterogeneous cluster, we use Ansible to login to each node, query that node’s characteristics, save the result, and then compiles those characteristics into a slurm configuration file, which can then be distributed throughout the cluster. We use Ansible extensively throughout our cluster to simplify management, and Slurm configuration is one of the best examples of where we use this effectively.

The following representative lines demonstrate the difference in some of the slurm configurations.

```
NodeName=mage01 RealMemory=1032001 TmpDisk=561217 CPUs
=80 Sockets=8 CoresPerSocket=10 ThreadsPerCore=1 Gres=
fabric:eth:no_consume:10,fabric:ib:no_consume:40,fabri
c:roce:no_consume:10, Feature=mages,acpi,aes,aperfmper
f,apic,arat,arch_perfmon,bts,clflush,cmov,constant_tsc
,cx16,cx8,dca,de,ds_cpl,dtes64,dtherm,dts,eagerfpu,epb
,ept,est,flexpriority,fpu,fxsr,ht,ibpb,ibrs,ida,lahf_l
m,lm,mca,mce,mmx,monitor,msr,mtrr,nonstop_tsc,nopl,nx,
pae,pat,pbe,pcid,pclmulqdq,pdcm,pdpe1gb,pebs,pge,pni,p
opcnt,pse,pse36,rdtscp,rep_good,sep,smx,ss,ssbd,sse,ss
e2,sse4_1,sse4_2,ssse3,syscall,tm,tm2,tpr_shadow,tsc,v
me,vmx,vnmi,vpid,x2apic,xtopology,xtpr,nogpu State=UNK
NOWN

NodeName=elf06 RealMemory=64336 TmpDisk=129777 CPUs=16
 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 Gres=fabr
ic:eth:no_consume:10,fabric:ib:no_consume:40,fabric:ro
ce:no_consume:10, Feature=elves-16c,elves,acpi,aes,ape
rfmperf,apic,arat,arch_perfmon,avx,bts,clflush,cmov,co
nstant_tsc,cx16,cx8,dca,de,ds_cpl,dtes64,dtherm,dts,ea
gerfpu,ept,est,flexpriority,fpu,fxsr,ht,ibpb,ibrs,ida,
lahf_lm,lm,mca,mce,mmx,monitor,msr,mtrr,nonstop_tsc,no
pl,nx,pae,pat,pbe,pcid,pclmulqdq,pdcm,pdpe1gb,pebs,pge
,pln,pni,popcnt,pse,pse36,pts,rdtscp,rep_good,sep,smx,
ss,ssbd,sse,sse2,sse4_1,sse4_2,ssse3,syscall,tm,tm2,tp
r_shadow,tsc,tsc_deadline_timer,vme,vmx,vnmi,vpid,x2ap
ic,xsave,xsaveopt,xtopology,xtpr,nogpu State=UNKNOWN

NodeName=elf77 RealMemory=96624 TmpDisk=98531 CPUs=20
Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 Gres=fabr
ic:eth:no_consume:10,fabric:ib:no_consume:40,fabric:ro
ce:no_consume:10, Feature=elves-20c,elves,acpi,aes,ape
rfmperf,apic,arat,arch_perfmon,avx,bts,clflush,cmov,co
nstant_tsc,cx16,cx8,dca,de,ds_cpl,dtes64,dtherm,dts,ea
gerfpu,ept,erms,est,f16c,flexpriority,fpu,fsgsbase,fxs
r,ht,ibpb,ibrs,ida,lahf_lm,lm,mca,mce,mmx,monitor,msr,
mtrr,nonstop_tsc,nopl,nx,pae,pat,pbe,pcid,pclmulqdq,pd
cm,pdpe1gb,pebs,pge,pln,pni,popcnt,pse,pse36,pts,rdran
d,rdtscp,rep_good,sep,smep,smx,ss,ssbd,sse,sse2,sse4_1
,sse4_2,ssse3,syscall,tm,tm2,tpr_shadow,tsc,tsc_deadli
ne_timer,vme,vmx,vnmi,vpid,x2apic,xsave,xsaveopt,xtopo
logy,xtpr,nogpu State=UNKNOWN

NodeName=hero34 RealMemory=128822 TmpDisk=770447 CPUs=
24 Sockets=2 CoresPerSocket=12 ThreadsPerCore=1 Gres=f
abric:eth:no_consume:40,fabric:roce:no_consume:40, Fea
ture=heroes,abm,acpi,aes,aperfmperf,apic,arat,arch_per
fmon,avx,avx2,bmi1,bmi2,bts,clflush,cmov,constant_tsc,
cqm,cqm_llc,cqm_occup_llc,cx16,cx8,dca,de,ds_cpl,dtes6
4,dtherm,dts,eagerfpu,epb,ept,erms,est,f16c,flexpriori
ty,fma,fpu,fsgsbase,fxsr,ht,ibpb,ibrs,ida,intel_ppin,i
nvpcid,lahf_lm,lm,mca,mce,mmx,monitor,movbe,msr,mtrr,n
onstop_tsc,nopl,nx,pae,pat,pbe,pcid,pclmulqdq,pdcm,pdp
e1gb,pebs,pge,pln,pni,popcnt,pse,pse36,pts,rdrand,rdts
cp,rep_good,sdbg,sep,smep,smx,ss,ssbd,sse,sse2,sse4_1,
sse4_2,ssse3,syscall,tm,tm2,tpr_shadow,tsc,tsc_adjust,
vme,vmx,vnmi,vpid,x2apic,xsave,xsaveopt,xtopology,xtpr
,nogpu State=UNKNOWN

NodeName=wizard01 RealMemory=95263 TmpDisk=803709 CPUs
=32 Sockets=2 CoresPerSocket=16 ThreadsPerCore=1 Gres=
fabric:eth:no_consume:10,fabric:opa:no_consume:100,gpu
:nvidia_geforce_gtx_1080_ti:4 Feature=wizards,3dnowpre
fetch,abm,acpi,adx,aes,aperfmperf,apic,arat,arch_perfm
on,art,avx,avx2,avx512bw,avx512cd,avx512dq,avx512f,avx
512vl,bmi1,bmi2,bts,cat_l3,cdp_l3,clflush,clflushopt,c
lwb,cmov,constant_tsc,cqm,cqm_llc,cqm_mbm_local,cqm_mb
m_total,cqm_occup_llc,cx16,cx8,dca,de,ds_cpl,dtes64,dt
herm,dts,eagerfpu,epb,ept,erms,est,f16c,flexpriority,f
ma,fpu,fsgsbase,fxsr,hle,ht,ibpb,ibrs,ida,intel_ppin,i
ntel_pt,intel_stibp,invpcid,lahf_lm,lm,mba,mca,mce,mmx
,monitor,movbe,mpx,msr,mtrr,nonstop_tsc,nopl,nx,ospke,
pae,pat,pbe,pcid,pclmulqdq,pdcm,pdpe1gb,pebs,pge,pku,p
ln,pni,popcnt,pse,pse36,pts,rdrand,rdseed,rdt_a,rdtscp
,rep_good,rtm,sdbg,sep,smap,smep,smx,spec_ctrl,ss,ssbd
,sse,sse2,sse4_1,sse4_2,ssse3,stibp,syscall,tm,tm2,tpr
_shadow,tsc,tsc_adjust,tsc_deadline_timer,vme,vmx,vnmi
,vpid,x2apic,xgetbv1,xsave,xsavec,xsaveopt,xtopology,x
tpr,gpu,gpu_nvidia State=UNKNOWN
```

These representative lines highlight

- The first line uses the oldest generation of CPUs in our cluster. Note that its feature set is a subset of the features of other CPUs.
- The second and third lines show that even with the same CPU architecture, we can differentiate on the number of cores and amount of memory.
- The fourth line is representative of our nodes which do not utilize a separate low-latency fabric, using RoCE only.
- The fifth line shows the capabilities of our newest CPU architecture, along with OmniPath and multiple GPUs.

Ansible will correctly detect and build a configuration file for nodes including

- CPU count
- Memory size
- Number of Sockets
- Cores per Socket
- Threads per Core
- Speed of Ethernet fabric
- Speed of Infiniband fabric, if present
- Speed of RoCE fabric, if present
- Speed of OmniPath fabric, if present
- CPU capabilities (e.g. avx2, avx512, bmi2)
- GPU model and number of GPUs

Even when identical compute nodes are purchased, many times they will be modified at a later time (e.g. a DIMM fails, or a GPU is added). Keeping these details manually would be quite difficult to manage, but by querying the hardware, Ansible can easily enumerate all of these variables.

Beocat runs on a fairly-common "condo" model, where individual researchers or research groups buy nodes on which they have priority access. If that research group is not using that node at any given time, that node's resources are available to the general community until needed again by that research group. A small number of nodes are entirely general access. We also generate these node configurations in Ansible to create a correct and consistent slurm.conf.

3.3 Operating System Consistency

To keep consistent updates, we generate a master OS (CentOS 7) boot image, which is booted via PXE or iPXE (not an uncommon practice among homogeneous clusters). Maintaining which systems can be booted from PXE, iPXE from an on-board card, or iPXE from a local EFI partition is still largely a manual process, as even minor configuration changes, especially on non-warranty hardware, can often change the boot method. Part of the boot process is to install local software, which is dependent on the installed hardware (e.g., nVidia GPU drivers).

When consistency cannot be achieved, we still standardize as much as possible. The single most powerful combination of tools, which helps us standardize, is Ansible and git. We use git to check-out and/or create the latest changes, and then use Ansible to push those changes everywhere. Aside from the compute system above, when deploying new hardware or virtual machines, we use Ansible to make the appropriate changes to DNS and/or DHCP, then install the OS, and finally use Ansible again to setup the entire environment, including how to communicate with the storage cluster, configuring monitoring, and enabling LDAP for logins and groups. Ansible is also used to configure our Cumulus Ethernet switches, with support for Lenovo switches promised soon. Even the switch configurations, which can't be pushed via Ansible, we still keep in git.

3.4 Software Optimization

Another powerful combination is EasyBuild and Lmod. While these tools were created for more traditional HPC clusters, we can invoke EasyBuild multiple times to optimize for each CPU architecture. This lets us make the most of our hardware, without having to compile all of our software for the lowest common denominator. Using EasyBuild, we currently optimize for four different classes (Westmere, Sandy Bridge/Ivy Bridge, Haswell/Broadwell, Skylake/Kaby Lake/Coffee Lake, and Epyc). The same Lmod command works on our entire compute infrastructure and will automatically load the most optimized version for the node on which it is run.

3.5 Storage Management

Not only has Beocat's compute infrastructure grown, the non-compute portion has grown as well. Anticipating this need, we implemented Ceph, and its posix-like filesystem, CephFS, as our primary filestore a few years ago, since it appears to be unparalleled in its ability to scale over time with a variety of underlying hardware. Expanding storage with Ceph consists of deploying hardware using the Ansible hardware deployment steps above, and then running a deployment script. Replacing individual hard drives is a similarly easy task. While this process would be the same as on a homogeneous cluster, the choice of Ceph over a larger hardware vendor was due to our constantly expanding footprint.

Even our Ceph cluster is heterogeneous. It began with 24 identical storage nodes, and has been expanded with some similar, but not identical hardware, and then expanded again with very different hardware. Currently our Ceph storage consists of (286) 6 TB hard drives, (95) 4 TB hard drives, (83) 8 TB hard drives, (64) 12 TB hard drives, (48) 400 GB NVMe cards, and (5) 480 GB Optane NVMe cards. In our experience, Ceph's own deployment tools have been inadequate to properly provision storage to our existing cluster. We wrote custom bash scripts to deploy, and in some cases re-deploy, hardware. Although these scripts work, the next time we have storage hardware to deploy, we will likely move these to Ansible and store them in git in order to standardize our deployment strategy and for ease-of-use for the future. As non-warranty storage hardware fails, we typically replace it with better hardware.

3.6 Non-HPC requirements

As our University's only dedicated research computing facility, we are frequently required to implement many related, but not-exactly-HPC workloads. To accomplish this, we have 7 virtualization servers running 65 virtual machines at the time of this writing. We leverage ganeti to manage and balance the workloads. Ganeti can use Ceph as an underlying block device. In a traditional larger cluster, another non-HPC group would handle these workloads. Using ganeti, we can deploy virtual servers in a very similar manner as we deploy raw hardware. We use ganeti VMs for our internal use (e.g., monitoring and login nodes), services we manage at the behest of other groups (e.g., web servers for research groups, remote execution hosts for research groups with node-locked licenses), and even services for which are completely managed by other groups, including our centralized IT.

3.7 Low-Latency Interconnects

One of our biggest continuing challenges, and one for which we do not have a good strategy, is the lack of consistency in low-latency interconnects. We have a mixture of Infiniband, RoCE (RDMA over converged Ethernet), and OmniPath. MPI jobs, if not specified correctly by the end-user, can be very slow. Replacing with one consistent interconnect is not cost effective at this point. This limits the number of nodes on which an MPI job can run effectively. This is a function of user education. Our application scientist constantly monitors running jobs to further optimize such jobs, which is a manual process and can be time-intensive.

3.8 Documentation

As a heterogeneous cluster, documentation is crucial. In a traditional HPC cluster, every rack looks essentially like every other rack. In Beocat, this is not the case. We make use of OpenDCIM to keep track of our assets and how all of our interconnects are configured. We use typical monitoring software (ganglia and icinga) to observe the system state, outages, and environmental factors. Any information, processes, and configurations, which cannot be captured by OpenDCIM, are kept in our Ansible git repositories or on our administrative wiki. OpenDCIM also produces some extensive maps of networks, as shown in Figure 1. The size of the maps are too extensive to view on a single page, but when viewed in a web browser, all the detail can be seen. Figure 2 shows a zoomed-in view of the same network map

All of our support documentation is also on a wiki. Critical sections (e.g., "Policies") are only able to be edited by administrators. Any Beocat user can edit any of the other public sections, and we encourage them to do so. The end users of the software have the domain knowledge that the system administrators do not. Many of them have developed non-obvious processes which can be related to others via the wiki. Admittedly, most users have been hesitant to edit the support wiki. Notably, we also have a separate non-public wiki for administrators to document processes, particularly for tasks which are performed only occasionally.

3.9 Communication

Similar to documentation, communication with end-users is vital to engage the research community. The tools and methods discussed here are not unique to heterogeneous clusters, but should be useful to any organization with a small number support staff.

3.9.1 *Support.*

Since near its inception, Beocat has relied on email for end-user support through Request Tracker (RT). RT ingests an email, creates a ticket, and sends an email to all those who can service the ticket. Replies can be either via email or via RT's web interface. By using such a ticketing system, end users can be assured that the support staff has received the support request, and the support staff has a record of all communications. The email interface means that support staff can filter emails to ensure those requests get answers, and easy answers can often be handled immediately, since emails can be responded to from phones or laptops. Additionally, we recently began instituting "open support hours," a time where any available support staff will be on-hand to answer questions in person. These open support hours are held in the student union, providing an unintimidating and comfortable place to meet.

3.9.2 *Announcements.*

For several years, we have had an email list, to which all current users of Beocat are required to subscribe. Over the past few years, we found that we were increasingly sending messages that were only pertinent to a small subset of users, usually only those who had currently running jobs or only those with jobs in the queue. Knowing that when you email people irrelevant information, they tend to ignore your emails before they even know if a particular message is germane to them, we sought to reduce the amount of traffic on the email list. The support email list is still used to announce major changes in software or hardware, opportunities to contribute to Beocat, and upcoming maintenance periods.

For messages which are more targeted, we turned to Twitter. For those end users who do not regularly use Twitter, our most recent tweets are featured on the front page of our support wiki. Increasingly, end users are already using Twitter anyway and these announcements are timely. Some examples of the types of announcements we have tweeted rather than emailed are

- There is an air conditioning outage in the data center. Currently running jobs have been suspended until the air conditioning has been fixed
- We have received reports that some people are experiencing long login times. We are investigating the cause.
- Our filesystems are really busy (not broken, just overloaded) right now, causing some slowdowns. We appreciate your patience.

3.10 Authentication

Our philosophy on identity management has been to avoid it as much as possible. In our case, this means that we run our own OpenLDAP server, but we offload all password management to our University's central LDAP services. While we realize that not all clusters will be able to take advantage of such an outside resource, it is a great help for those that can. We never have to manage password resets, expirations, etc. The process we use is:

- The user creates an account via the university's webpage. Note that anybody can create an account, as this is the first step a potential student must take to apply for admission. The creation of an account does not give them any privileges anywhere. It simply serves as a central authority for identity management.
- The user must apply for a Beocat account via our own web site. It includes information we require for reporting purposes, such as demographics, as well as host institution and department. The account request is sent as an email to the HPC director and system administrators. We built an account management site quickly using Pyramid (https://trypyramid.com/).
- The account is approved (or, in rare cases, denied) by the director of our HPC site via a reply to the email.
- Upon approval, the system administrators use the same website to initiate the account provisioning process. Everything is automated from this point forward. LDAP accounts are setup, home directories are created, and slurm permissions are generated.
- Slurm projects are managed through the same web interface.

3.11 Internally-Developed Software

Finally, we have two programs which have been developed in-house, which help us to administer the cluster.

'kstat' is a Perl program which aggregates information from many of slurm's built-in tools. Administrators use kstat to monitor the job queue and usage. Most users use kstat to view where their jobs are in the queue and how long they have been running. Advanced users use kstat to monitor their own jobs and use that information to optimize their use on future runs. kstat could be used on a cluster of any size. kstat source is available at https://gitlab.beocat.ksu.edu/Admin-Public/kstat .

Figure 3 shows some representative lines from kstat.T he first part of the output lists all nodes (truncated for display here) and which jobs are running there, as well as how many cores, how much memory, and which project group (if any) owns that node. Lines below the green line (again, truncated here) show jobs waiting in the queue, along with some key job parameters and the time they have been in the queue. All colors have a representative meaning. For example, text with red background indicates something wrong. In this case, we have one node (mole039) which is not responding and a job which has been waiting for a very long time. Troubleshooting these indicated that mole039 had a hardware error and the user that had been waiting a long time had incorrectly specified some job parameters.

'NetPIPE' (Network Protocol-Independent Performance Evaluater) is a program originally developed by Ames Laboratory and Iowa State University, and is currently maintained by one of this paper's authors. It has been invaluable in evaluating and comparing low-latency fabrics and pinpointing performance problems. More information about NetPIPE as well as its source code are available at http://netpipe.cs.ksu.edu/.

4 EVALUATION

Since moving to slurm in January of 2018 until the time of this writing, approximately 16 months later, Beocat has serviced well over 3 million jobs, accounting for over 50 million CPU-hours. New nodes are added two to three times per year. With the same CPU architecture, nodes can be running jobs typically within 30 minutes of physical installation. With new CPU architectures added, nodes can be brought online within 2-3 days after physical installation. All downtime has been due to physical considerations (e.g., datacenter air conditioning) and planned maintenance. We have experienced some slower performance than expected during this time, but have no metrics on those.

Except during the busiest times, "small" jobs (fewer than 4 cores, fewer than 4 GB of RAM per core, and a runtime of less than 24 hours) start within minutes. Nearly all jobs of less than 400 cores begin within a week of submission.

5 SUMMARY

Just because a cluster is dynamic and heterogeneous does not mean an exponential amount of work needs to go into its management. The effective use of existing software can greatly ease the burden of a system administrator, save a lot of time when deploying new hardware, and simultaneously increase end-user satisfaction. While we make no claims of perfection, we do believe that over the course of time, the systems implemented on Beocat can be a good example for growing and heterogeneous HPC clusters to follow.

6 ACKNOWLEDGEMENTS

This paper details the Beocat Research Cluster at Kansas State University, which is funded in part by NSF grants CNS-1006860, EPS-1006860, EPS-0919443, ACI-1440548, CHE-1726332, and NIH P20GM113109.

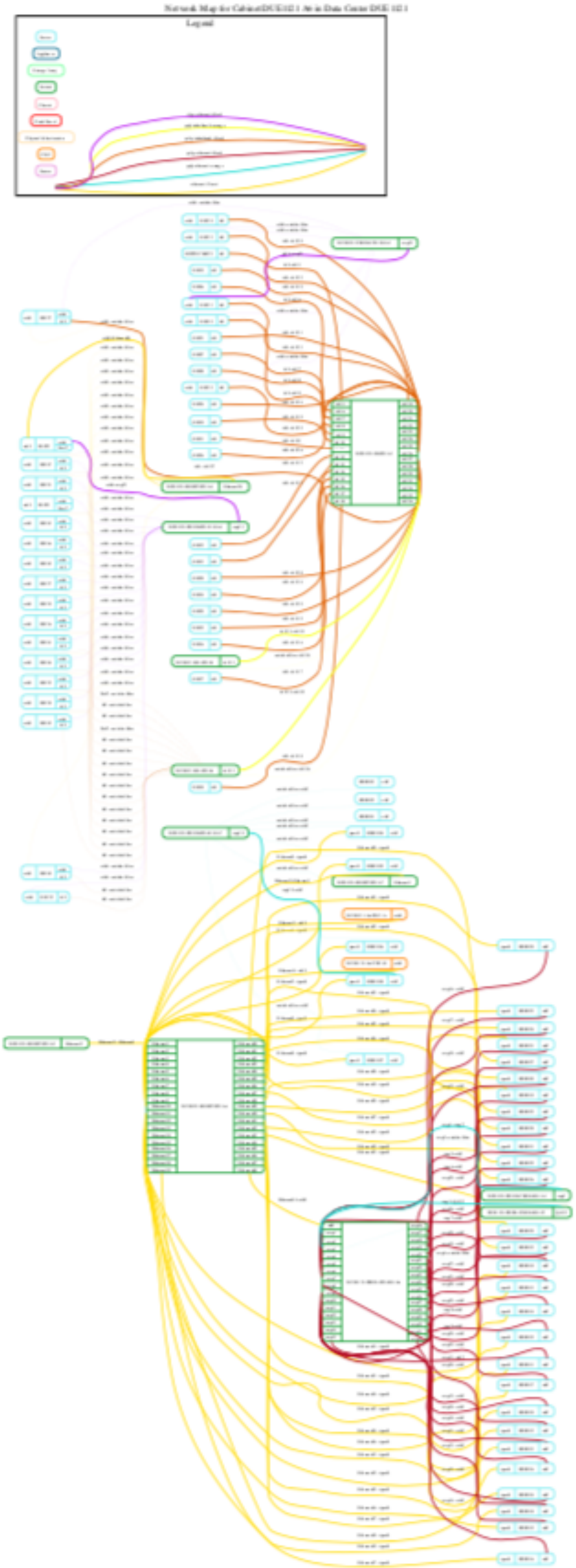


Figure 1: OpenDCIM network map of a single rack in Beocat. Each network fabric or type has its own color. Colors can be chosen or randomly selected.

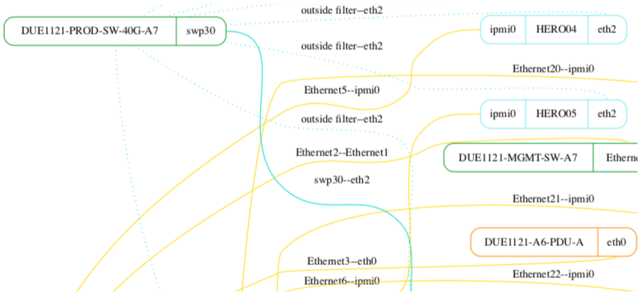
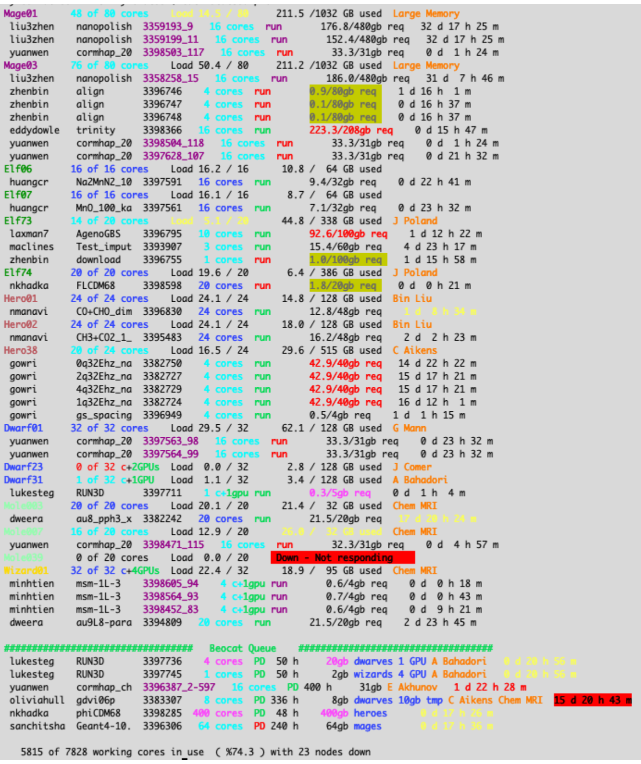


Figure 2: Zoomed in view of part of Figure 1



```
Mage01      48 of 80 cores    Load 14.5 / 80     211.5 /1032 GB used  Large Memory
 liu3zhen    nanopolish  3359193_9    16 cores  run       176.8/480gb req   32 d 17 h 25 m
 liu3zhen    nanopolish  3359199_11    16 cores  run       152.4/480gb req   32 d 17 h 25 m
 yuanwen     cormhap_20  3398503_117    16 cores  run       33.3/31gb req    0 d  1 h 24 m
Mage03      76 of 80 cores    Load 50.4 / 80     211.2 /1032 GB used  Large Memory
 liu3zhen    nanopolish  3358258_15    16 cores  run       186.0/480gb req   31 d  7 h 46 m
 zhenbin     align       3396746     4 cores  run        0.9/80gb req    1 d 16 h  1 m
 zhenbin     align       3396747     4 cores  run        0.1/80gb req    0 d 16 h 37 m
 zhenbin     align       3396748     4 cores  run        0.1/80gb req    0 d 16 h 37 m
 eddydowle   trinity     3398366    16 cores  run        223.3/208gb req    0 d 15 h 47 m
 yuanwen     cormhap_20  3398504_118    16 cores  run       33.3/31gb req    0 d  1 h 24 m
 yuanwen     cormhap_20  3397628_107    16 cores  run       33.3/31gb req    0 d 21 h 32 m
Elf06       16 of 16 cores    Load 16.2 / 16      10.8 /  64 GB used
 huangcr     Na2MnN2_10  3397591    16 cores  run        9.4/32gb req    0 d 22 h 41 m
Elf07       16 of 16 cores    Load 16.1 / 16       8.7 /  64 GB used
 huangcr     MnO_100_ka  3397561    16 cores  run        7.1/32gb req    0 d 23 h 32 m
Elf73       14 of 20 cores    Load  5.1 / 20      44.8 / 338 GB used  J Poland
 laxman7     AgenoGBS    3396795    10 cores  run        92.6/100gb req    1 d 12 h 22 m
 maclines    Test_imput  3393907     3 cores  run        15.4/60gb req    4 d 23 h 17 m
 zhenbin     download    3396755     1 cores  run        1.0/100gb req    1 d 15 h 58 m
Elf74       20 of 20 cores    Load 19.6 / 20       6.4 / 386 GB used  J Poland
 nkhadka     FLCDM68     3398598    20 cores  run        1.8/20gb req    0 d  0 h 21 m
Hero01      24 of 24 cores    Load 24.1 / 24      14.8 / 128 GB used  Bin Liu
 nmanavi     CO+CHO_dim  3396830    24 cores  run        12.8/48gb req     1 d  8 h 34 m
Hero02      24 of 24 cores    Load 24.1 / 24      18.0 / 128 GB used  Bin Liu
 nmanavi     CH3+CO2_1_  3395483    24 cores  run        16.2/48gb req    2 d  2 h 23 m
Hero38      20 of 24 cores    Load 16.5 / 24      29.6 / 515 GB used  C Aikens
 gowri       0q32Ehz_na  3382750     4 cores  run        42.9/40gb req   14 d 22 h 22 m
 gowri       2q32Ehz_na  3382727     4 cores  run        42.9/40gb req   15 d 17 h 21 m
 gowri       4q32Ehz_na  3382729     4 cores  run        42.9/40gb req   15 d 17 h 21 m
 gowri       1q32Ehz_na  3382724     4 cores  run        42.9/40gb req   16 d 12 h  1 m
 gowri       gs_spacing  3396949     4 cores  run        0.5/4gb req    1 d  1 h 15 m
Dwarf01     32 of 32 cores    Load 29.5 / 32      62.1 / 128 GB used  G Mann
 yuanwen     cormhap_20  3397563_98    16 cores  run       33.3/31gb req    0 d 23 h 32 m
 yuanwen     cormhap_20  3397564_99    16 cores  run       33.3/31gb req    0 d 23 h 32 m
Dwarf23     0 of 32 c+2GPUs   Load  0.0 / 32       2.8 / 128 GB used  J Comer
Dwarf31     1 of 32 c+1GPU    Load  1.1 / 32       3.4 / 128 GB used  A Bahadori
 lukesteg    RUN3D       3397711     1 c+1gpu run        0.3/5gb req    0 d  1 h  4 m
Mole003     20 of 20 cores    Load 20.1 / 20      21.4 /  32 GB used  Chem MRI
 dweera      au8_pph3_x  3382242    20 cores  run        21.5/20gb req    17 d 20 h 24 m
Mole007     16 of 20 cores    Load 12.9 / 20      26.0 /  32 GB used  Chem MRI
 yuanwen     cormhap_20  3398471_115    16 cores  run       32.3/31gb req    0 d  4 h 57 m
Mole039     0 of 20 cores     Load  0.0 / 20     Down - Not responding
Wizard01    32 of 32 c+4GPUs  Load 22.4 / 32      18.9 /  95 GB used  Chem MRI
 minhtien    msm-1L-3    3398605_94    4 c+1gpu run       0.6/4gb req    0 d  0 h 18 m
 minhtien    msm-1L-3    3398564_93    4 c+1gpu run       0.7/4gb req    0 d  0 h 43 m
 minhtien    msm-1L-3    3398452_83    4 c+1gpu run       0.6/4gb req    0 d  9 h 21 m
 dweera      au9L8-para  3394809    20 cores  run        21.5/20gb req    2 d 23 h 45 m

###################################   Beocat Queue    ####################################
 lukesteg    RUN3D       3397736     4 cores  PD  50 h      20gb dwarves 1 GPU A Bahadori    0 d 20 h 56 m
 lukesteg    RUN3D       3397745     1 cores  PD  50 h       2gb wizards 4 GPU A Bahadori    0 d 20 h 56 m
 yuanwen     cormhap_ch  3396387_2-597    16 cores  PD 400 h     31gb E Akhunov    1 d 22 h 28 m
 oliviahull  gdvi06p     3383307     8 cores  PD 336 h      8gb dwarves 10gb tmp C Aikens Chem MRI   15 d 20 h 43 m
 nkhadka     phiCDM68    3398285   400 cores  PD  48 h    400gb heroes         0 d 17 h 26 m
 sanchitsha  Geant4-10.  3396306    64 cores  PD 240 h     64gb mages          0 d 17 h 36 m

   5815 of 7828 working cores in_use  ( %74.3 ) with 23 nodes down
```

Figure 3: kstat sample output (truncated)