SRCC

STANFORD RESEARCH COMPUTING CENTER

slurm on Sherlock

workload manager

Slurm User Group 2019

17-18 September 2019 - Salt Lake City, Utah, USA

Kilian Cavalotti

Stanford Research Computing Center

Hi!

My name is **Kilian** Cavalotti

I'm HPC Technical Lead & Architect at Stanford University

I manage **Sherlock**, the Stanford shared HPC cluster

Stanford Research Computing Center

Our mission

Build & support a comprehensive program and capabilities to advance **computational** and **data-intensive** research at Stanford

Sherlock is

- a **shared** HPC cluster, operated by the Stanford Research Computing Center
- available at **no cost** to all Stanford Faculty members and their research teams to **support sponsored research**
- a **condo** cluster, where PIs can become ***owners***
  - they get their own Slurm partition
  - they get access to all the other owner's nodes when they're not in use (owner jobs preempt background jobs)

A little bit of **history**

2014 | Sherlock opens for production
initial seed of 120 nodes, funded by the University Provost

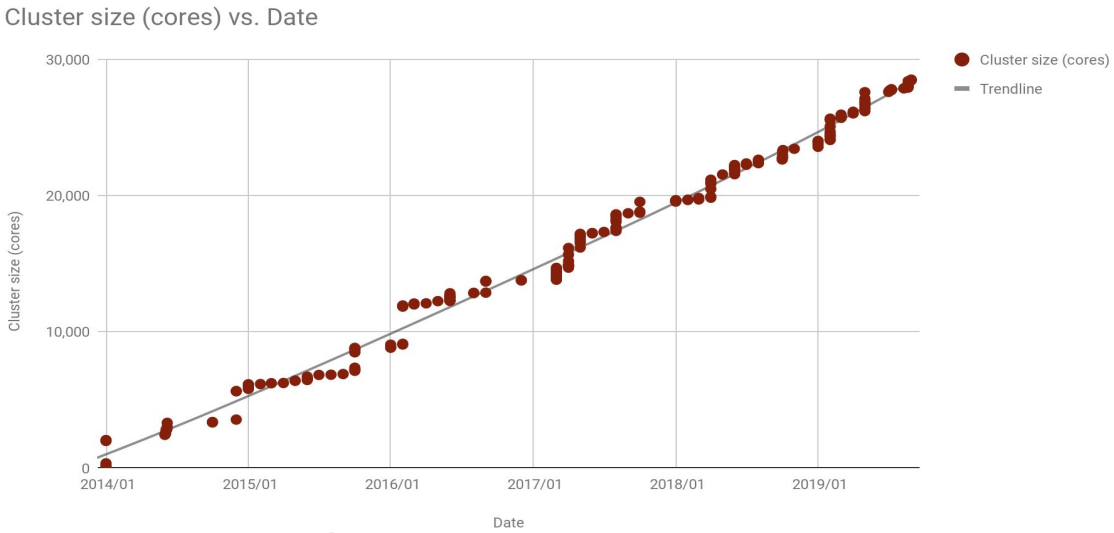
2016 | Sherlock reaches capacity
FDR Infiniband fabric maxed out, ~800 PI-owned nodes

2017 | Sherlock 2.0
complete hardware, platform, software and services refresh

2019 | Sherlock 2.0 reaches capacity
EDR IB fabric maxed out

2020 | Sherlock 3.0
hardware refresh, HDR IB fabric rollout

Successful model



- 15x growth in 5 years
- >90% of Sherlock's nodes are owners nodes

Sherlock

as of today, things change on a weekly basis

- **1,502** compute nodes
  16 → 64 CPU cores, 64 → 3TB RAM
  from Ivy Bridge to Skylake
- **780** GPUs
  from K20s to V100 SXM2 & TITANs
  4x or 8x GPUs per node
- **2** IB fabrics
  2:1 FDR + 2:1 EDR
- **Slurm** 18.08.8
  CentOS 7.6

Overall

29,192 CPU cores

6 PB scratch + 12 PB long-term storage

>4,500 users

from all the 7 schools, SLAC, Stanford institutes, etc.
from astrophysics to the music department

>720 PI groups, 125 owner groups

over 3,500 support tickets per year

And more...

2.12 PFlops

FP64, FWIW, YMMV

46 racks

450 kW

96 IB switches

7,248 IB cables

17 server models

4 CPU/GPU generations

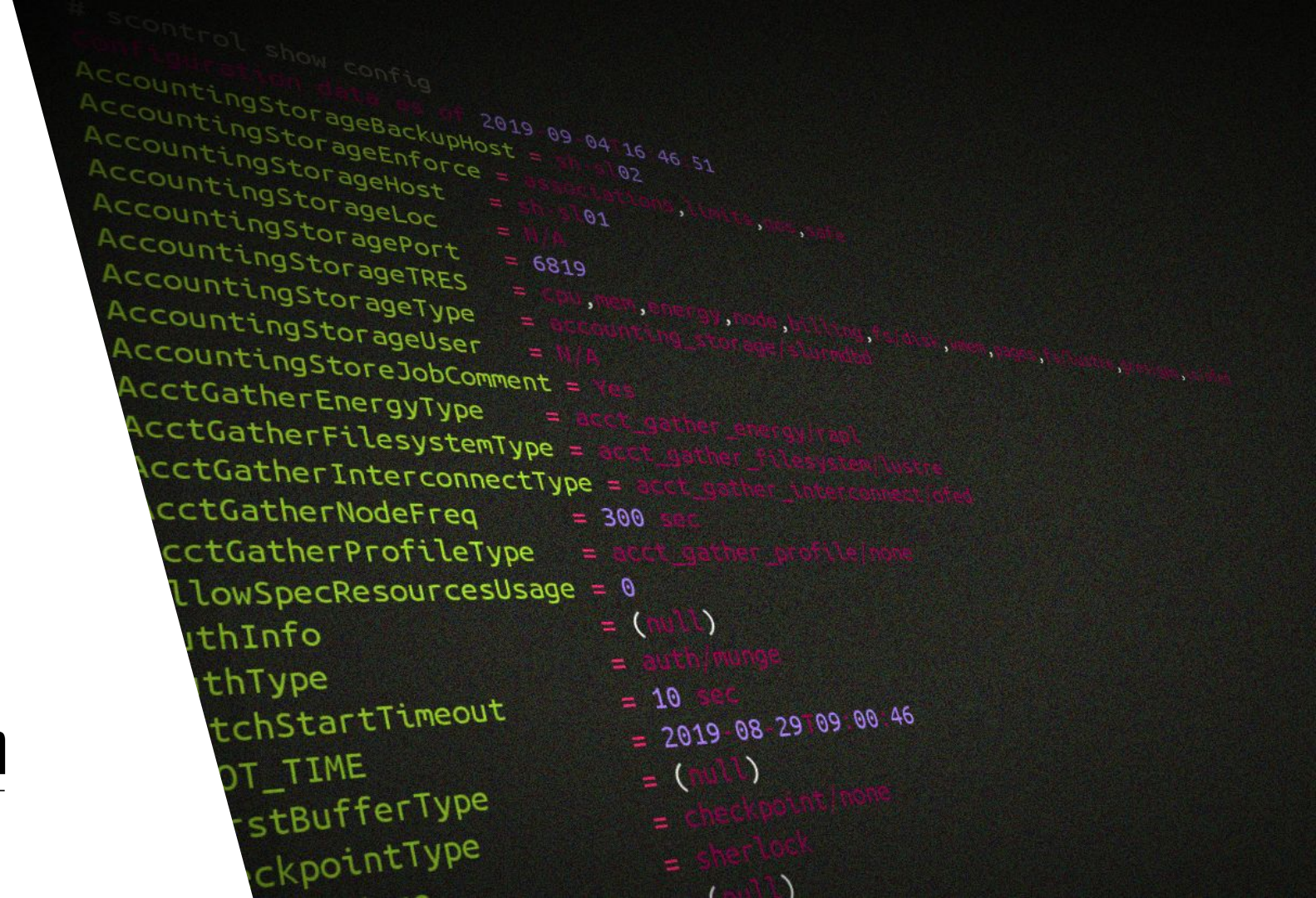
117 Slurm partitions

100,000 jobs/day

peaks at 250 jobs/s

15 million jobs since Jan 01

job #50,000,000 submitted on Sep. 10



slurm
workload manager

Slurm **partitions**

- Public partitions

  Anybody can use

  - `normal`: regular CPU nodes
  - `gpu`: GPU nodes
  - `bigmem`: large memory nodes
  - `dev`: interactive/debug nodes

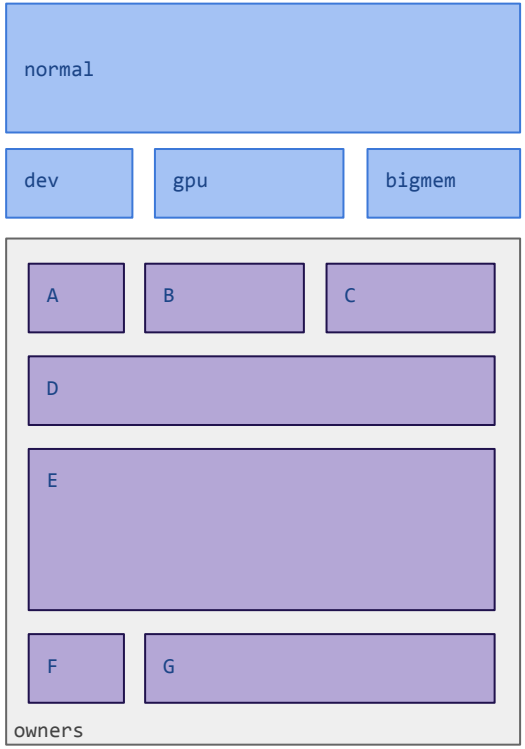
- Owner partitions

  Nodes purchased by PIs, one partition per owner

- `Owners` partition

  Preemptable partition for background owners' jobs

Partition **structure**



"Public" partitions

- All users can submit jobs there

Owner partitions

- PI A (only) can submit to partition "A"
- PI A can submit to "owners" and run on nodes from PI B
- When PI B submits to "B", PI A's job on B is preempted

Triggers

https://slurm.schedmd.com/strigger.html

triggers are generated when specific Slurm events occur, and can be used to run actions (scripts)

- events could be on the controller, the database daemon, jobs, nodes…
- we use triggers… a lot!
  - for notifications and monitoring
  - for automatic recovery actions (w/ NHC)

Triggers

```
RES_TYPE   RES_ID TYPE                                 OFFSET USER     FLAGS PROGRAM
node            * down                                     0 slurm    PERM  down.sh
node            * drained                                  0 slurm    PERM  drained.sh
node            * up                                       0 slurm    PERM  up.sh
slurmctld       * primary_slurmctld_resumed_operation      0 slurm    PERM  primary_slurmctld_resumed_op.sh
slurmctld       * primary_slurmctld_resumed_control        0 slurm    PERM  primary_slurmctld_resumed_control.sh
slurmctld       * backup_slurmctld_resumed_operation       0 slurm    PERM  backup_slurmctld_resumed_operation.sh
database        * primary_database_resumed_operation       0 slurm    PERM  primary_database_resumed_operation.sh
slurmdbd        * primary_slurmdbd_resumed_operation       0 slurm    PERM  primary_slurmdbd_resumed_operation.sh
slurmdbd        * primary_slurmdbd_failure                 0 slurm    PERM  primary_slurmdbd_failure.sh
slurmctld       * backup_slurmctld_failure                 0 slurm    PERM  backup_slurmctld_failure.sh
slurmctld       * primary_slurmctld_failure                0 slurm    PERM  primary_slurmctld_failure.sh
slurmctld       * backup_slurmctld_assumed_control         0 slurm    PERM  backup_slurmctld_assumed_control.sh
database        * primary_database_failure                 0 slurm    PERM  primary_database_failure.sh
slurmctld       * primary_slurmctld_acct_buffer_full       0 slurm    PERM  primary_slurmctld_acct_buffer_full.sh
```

When problems happen on nodes:

- NHC detects the issue, drains nodes
- script associated w/ events is triggered
- sysadmins are notified
- corrective actions are taken

Triggers

```
RES_TYPE   RES_ID TYPE
node            * down
node            * drained
node            * up
slurmctld       * primary_slurmctld_resumed_operation
slurmctld       * primary_slurmctld_resumed_control
slurmctld       * backup_slurmctld_resumed_operation
database        * primary_database_resumed_operation
slurmdbd        * primary_slurmdbd_resumed_operation
slurmdbd        * primary_slurmdbd_failure
slurmctld       * backup_slurmctld_failure
slurmctld       * primary_slurmctld_failure
slurmctld       * backup_slurmctld_assumed_control
database        * primary_database_failure
slurmctld       * primary_slurmctld_acct_buffer_full
```

On `slurmctld` / `slurmdb` events:

- sysadmins are notified
- events are recorded

Node weights

- get the best performance out of the box
  - provide the highest-end CPU when possible
- don't waste specialized resources
  - when not explicitly requested, allocate specialty nodes in last resort
- encode node characteristics in node weights
  - nodes with lower weight are selected first
  - more memory / GRES = higher weight, so nodes are selected last
  - more recent CPUs = lower weight, give the best performance by default

Weight mask

```
#   Weight mask: 1 | #GRES | Memory | #Cores | CPUgen | 1
#       prefix is to avoid octal conversion
#       suffix is to avoid having null weights
#
#   Values:
#       #GRES   none: 0   Memory   64 GB: 0   #Cores  16: 0  CPUgen  ???: 3
#              1 GPU: 1            96 GB: 1           20: 1          CSL: 4
#              2 GPU: 2           128 GB: 2           24: 2          SKX: 5
#              3 GPU: 3           256 GB: 3           28: 3          BDW: 6
[...]

# BDW | 20c | 128GB
NodeName=[...] \
    Sockets=2 CoresPerSocket=10 \
    RealMemory=128000 \
    Weight=102161 \
    Feature="CPU_MNF:INTEL,CPU_GEN:BDW,CPU_SKU:E5-2640v4,CPU_FRQ:2.40GHz"
```

- Example: 20-core Broadwell w/ 128GB RAM, no GPU

  `Weight=102161`

Recurrent jobs

- user: “*can I do cron?*”
- sysadmin: “*nope, but you can Slurm it!*”

	Cron job	Recurrent job
Dedicated resources for the task	❌	✔
Persistent across node reinstallations	❌	✔
Unique, controlled execution	❌	✔
Precise schedule	✔	❌

Recurrent jobs

```
#!/bin/bash
#SBATCH --job-name=cron
#SBATCH --begin=now+7days
#SBATCH --dependency=singleton
#SBATCH --time=00:10:00
#SBATCH --mail-type=FAIL

./my_cron_script

## Resubmit the job for the next execution
sbatch $0
```

- weekly execution
  - execution deferred by at least 7 days
- ensure unique running instance
- email notification on failure
- works great for backup, data sync jobs

Persistent jobs

- user: “*can I have a database server?*”
- sysadmin: “*nope, but you can Slurm it!*”

- run untils explicitly `scancel`'ed
- signal itself before end of allocated time
- resubmit itself on termination
- re-runs right away (maybe)
  - more adapted to partitions with lower wait times
- persistent `$JOBID`!

Persistent jobs

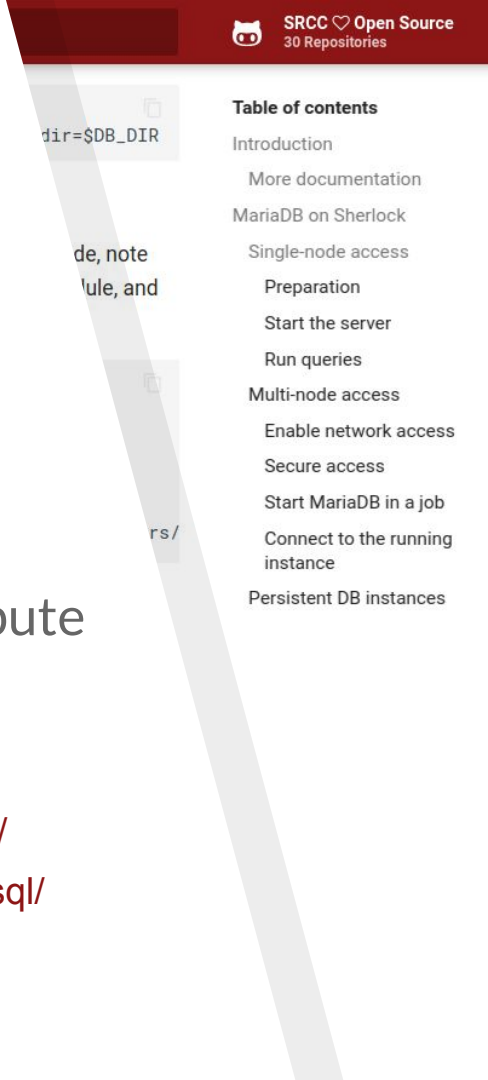
```
#!/bin/bash
#SBATCH --job-name=persistent
#SBATCH --dependency=singleton
#SBATCH --time=24:00:00
#SBATCH --signal=B:SIGUSR1@90

# catch the SIGUSR1 signal
_resubmit() {
    echo "$(date): job $SLURM_JOBID received SIGUSR1 at $(date), re-queueing"
    scontrol requeue $SLURM_JOBID
}
trap _resubmit SIGUSR1

./my_app &
wait
```

- signal `SIGUSR1` trapped by the `sbatch` step
  - main app needs to be running in the background for signal to be trapped
- `scontrol requeue` ensures persistent `$JOBID`
  - for easier job dependencies
- works great for database servers (MySQL, PostgreSQL...)

Wait, a database server in a job?

Sure, why not?

- resources are guaranteed and limited
- database service can run with user privileges
- db service jobs can be relocated to different compute nodes based on availability
- complete step-by-step instructions at
  - https://www.sherlock.stanford.edu/docs/software/using/mariadb/
  - https://www.sherlock.stanford.edu/docs/software/using/postgresql/



Command **caching**

- user: “*are my jobs running? Let's* `watch -n.1 squeue`”
- `slurmctld`: “*aaargh!*”

```
# Pseudo shell
_cache_cmd() { # $1: cmd, $2: cache lifetime (sec)
    if cached_output is still valid; then
        echo "$cached_output"
    else
        execute $1 and cache output for $2 seconds
    fi
}

# define command alias
squeue() { _cache_cmd /usr/bin/squeue 10 "$@"; }
```

- goes in users' profile, and cache `squeue` results for 10s
- works for all status commands (`sstat`, `sacct`, `sinfo`)

Job script **archiving**

- user: “*my job failed! why?*”
- sysadmin: “*what does your submission script look like?*”
- user: “*uuuh...*”

- we archive job scripts
  - in `PrologSlurmctld`

```
SLURM_SPOOLDIR="/var/spool/slurm.state"
SLURM_JOBSTORE="/share/admin/logs/slurm/jobs"

jobid=${SLURM_ARRAY_JOB_ID:-$SLURM_JOBID}
j_hsh=${jobid: -1}
cp $SLURM_SPOOLDIR/hash.${jobid: -1}/job.$jobid/environment $JOB_STORE/env
cp $SLURM_SPOOLDIR/hash.${jobid: -1}/job.$jobid/script      $JOB_STORE/script
```

Job submit plugin

- for a long time: *we don't need no job submit plugin!*
- nowadays: *how did we ever worked without it?*
- we use it to:
  - automatically assign licenses to jobs
  - drop un-authorized options (`--reboot`, `--exclusive` in shared partitions)
  - provide helpful messages to users (when their job is rejected)

```
[kilian@sh-ln04 login ~]$ srun -p gpu --pty bash
srun: error: ================================================================================
srun: error:  ERROR: missing GPU request, job not submitted
srun: error: ================================================================================
srun: error:  Jobs submitted to the gpu partition must explicitly request GPUs, by using
srun: error:  the --gres option.
srun: error: --------------------------------------------------------------------------------
srun: error: QOSMinGRES
srun: error: Unable to allocate resources: Job violates accounting/QOS policy (job submit
limit, user's size and/or time limits)
```

GPU mode SPANK plugin

- GPU compute modes
  - `EXCLUSIVE_PROCESS` by default, good for most cases
  - but some applications *need* multiple contexts on GPU
  - changing GPU compute mode requires `root`
- https://github.com/stanford-rc/slurm-spank-gpu_cmode
  - SPANK plugin to let users specify the GPU compute mode they need

```
$ srun --help
      --gpu_cmode=<shared|exclusive|prohibited>
                  Set the GPU compute mode on the allocated GPUs to
                  shared, exclusive or prohibited. Default is
                  Exclusive

$ srun --gres gpu:1 --gpu_cmode=shared "nvidia-smi --query-gpu=compute_mode --format=csv,noheader"
Default
$ srun --gres gpu:1 --gpu_cmode=exclusive nvidia-smi --query-gpu=compute_mode --format=csv,noheader
Exclusive_Process
```

Expected **wait times**

- user: “*how long will my job wait?*”
- sysadmin: “*why don't we ask the scheduler?*”

```
$ ssh sherlock
[...]
--------------------------------------------------------------------------------
Sherlock    status   | OPERATIONAL     | uptime : 99.989%
            usage    | normal:  98.44% | use/tot:  1,764/ 1,792 cores
                     | global:  93.66% | use/tot: 26,732/28,540 cores

kilian      cur.jobs | 0 RUNNING (0 core), 0 PENDING (0 core)
            job wait | 16 hours and 7 minutes in normal
--------------------------------------------------------------------------------
```

- profile runs `srun --test-only` at login
  - gives a user-specific estimation of typical job wait time, including fairshare
  - consumes one jobid, but ¯_(ツ)_/¯

Slurm dashboards

- overall usage
- scheduler internals
- queue info
- nodes states
- node utilization
- partition usage

Running

Running jobs	Running CPUs	Running GPUs	Active Nodes	Active partitions
4594	25405	202	1377	68

Down

Completing nodes	Completing jobs	Down nodes
2	2	15

Waiting

Pending jobs	Pending CPUs	Pending GPUs
8226	73855	707

Idle

Idle nodes	Idle CPUs	Idle GPUs	Idle partitions
9	1403	513	46

Running CPUs (all)

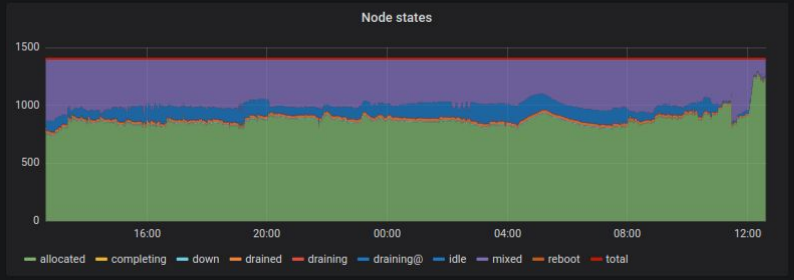
95%

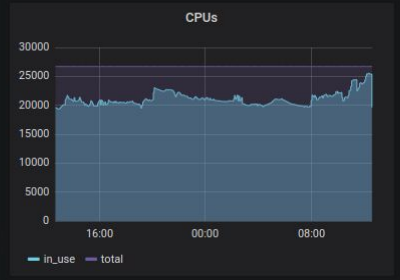
Running CPUs (normal)

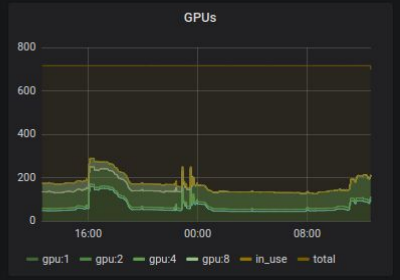
97%

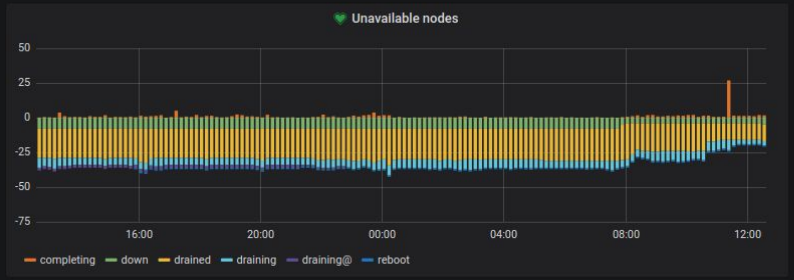
Running CPUs (owners)

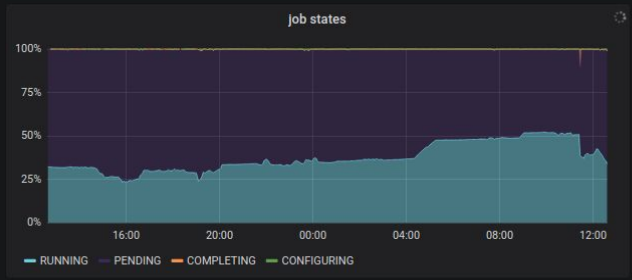
96%

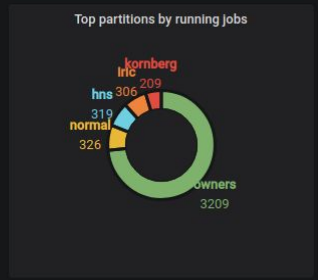


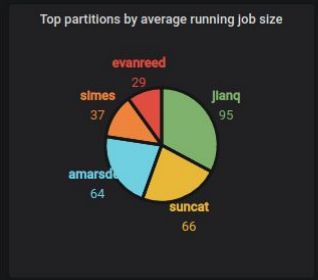


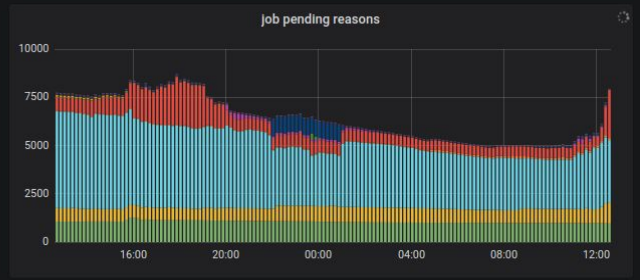








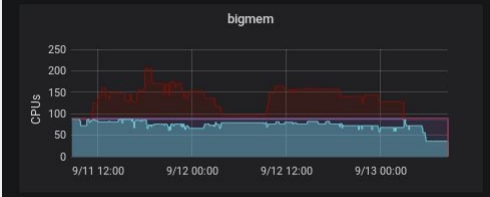


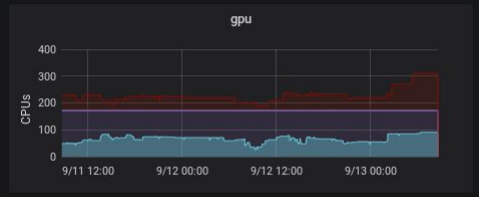


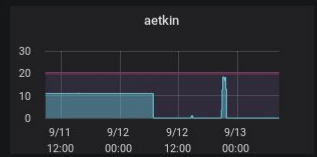

Sherlock 2.0 Slurm - partitions live activity
Last 24 hours
1m
normal
89%
gpu
44%
dev
24%
bigmem
90%
aaiken 0%
aboettig N/A
aetkin 51%
agitler 0%
akundaje 86%
amarsden 65%
andrewg 44%
andyhall 15%
arend 20%
astraigh 43%
athey 8%
awagner 0%
baccus N/A
biochem 67%
brunger 86%
candes 61%
casco 0%
cbohon 0%
cee 75%
cees 31%
cgawad N/A
cmackall 0%
deho 62%
delp 0%
diffenbaugh 0%
djames 0%
donoho 5%
dpetrov 29%
dpwall 49%
drehkopf N/A
dschneid 0%
egyang 0%
eriking 40%
euan 20%
evanreed 31%
fejer 0%
fischbac 0%
fkessler 0%
fmcnich 0%
gentzkow 84%
giocomo 31%
greicius 0%
gross 0%
haiwang 55%
hbfraser 19%
hlwill 36%
hns 58%
horence 41%
howchang 41%
iang 8%
ibiis 23%
iric 89%
isoltesz 0%
jamesz 0%
jduchi 0%
jgreiter 0%
jhyoon1 0%
jianq 81%
jonfan 0%
jp1 0%
jpriest 26%
kcgarcia 46%
khavari 19%
klposton 0%
kobilka 44%
konings 0%
kornberg 55%
ljinhy 65%
lmackey 0%
maggiori 0%
mc 2%
mcovert 84%
menon 25%
mfeldman 0%
mignot 46%
mrivas 31%
msalit 0%
mschnitz 0%
mvperez N/A
mzeineh 0%
nano 0%
nbloom 5%
oro 56%
pande 0%
pelc 93%
pfordyce N/A
phn 42%
possu 17%
pritch 15%
ptass 93%
puglisi 84%
qsu 79%
quake 80%
rbaltman 52%
rondror 17%
russpold 0%
rzia 32%
schumer 0%
serc 23%
sfgf 20%
shenoy 0%
simes 80%
skachru 93%
slqi 0%
spalumbi 0%
stat 0%
stuartkm 0%
suncat 11%
syyeung 0%
trc 0%
whwong 61%
willhies 0%
wjg 32%
wolak 93%
yamins 19%
yiorgo 37%
owners
37%

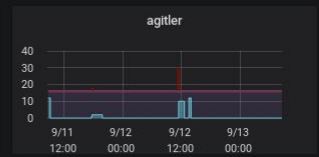
Sherlock Sherlock 2.0 Slurm - partitions historical activity

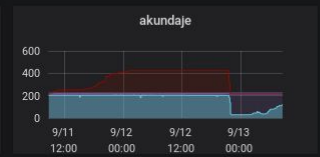
Last 2 days

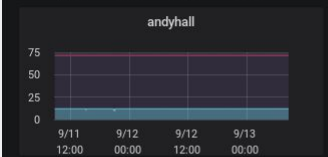


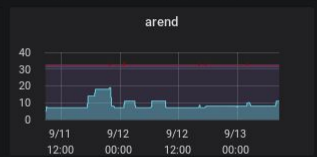


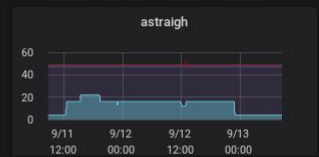


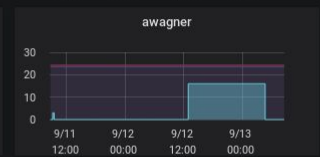


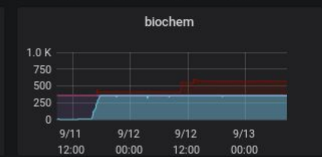


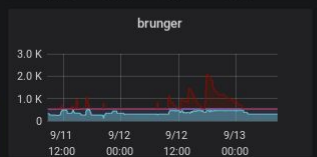


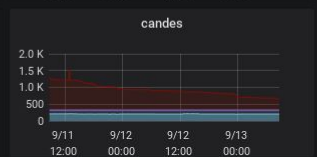


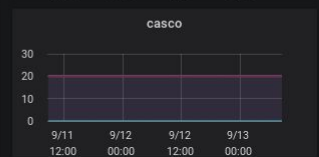


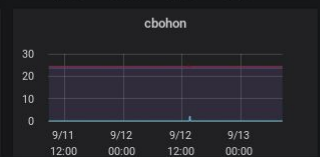


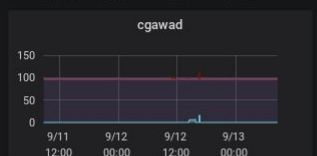


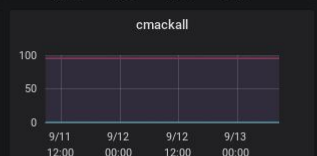


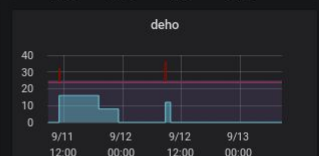


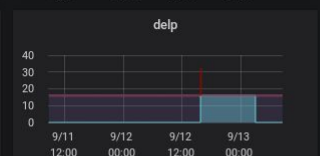


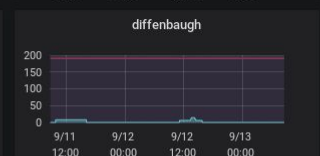


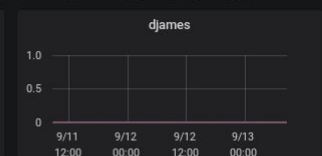


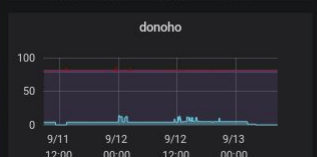


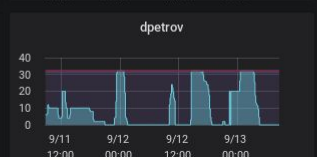


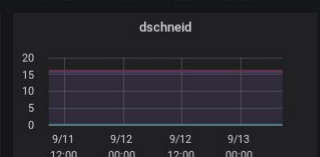


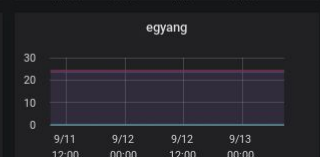


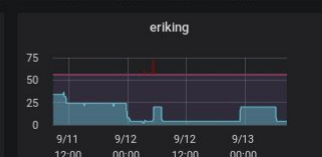










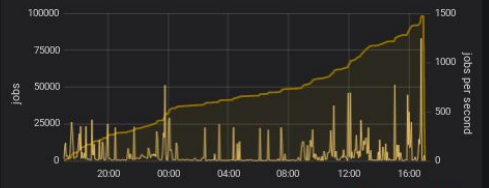


Sherlock 2.0 Slurm - internals

Last 24 hours

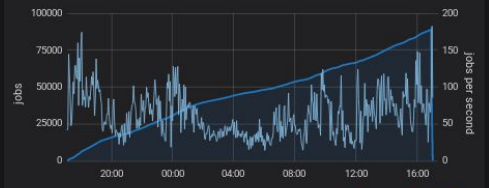
Jobs

jobs submitted



	max	avg	current
jobs submitted	98014	44915	60
jobs submitted per second (right-y)	1244	68	3

jobs started

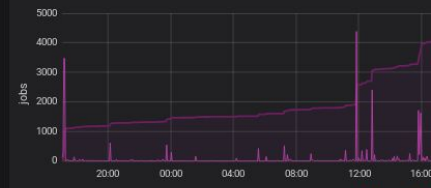


	max	avg	current
jobs started	88931	45607	296
jobs started per second (right-y)	183	62	86

jobs completed

	max	avg	current
jobs completed	80379	41474	150
jobs completed per second (right-y)	159	56	32

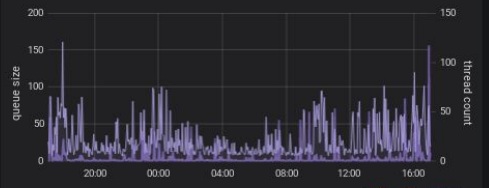
jobs cancelled



	max	avg
jobs canceled	4144	1853
jobs canceled per second (right-y)	219	3

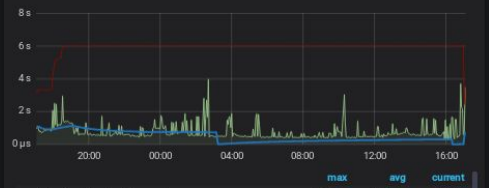
Scheduler

server threads / agent queue



	max	avg	current
agent queue size	155	4	11
server thread count (right-y)	121	21	15

last cycle time



	max	avg	current
time for last scheduling cycle	3.96 s	762 ms	3.47 s
scheduling cycle mean time	1.12 s	472 ms	745 ms
scheduling cycle max time	6.01 s	5.87 s	3.53 s

cycles

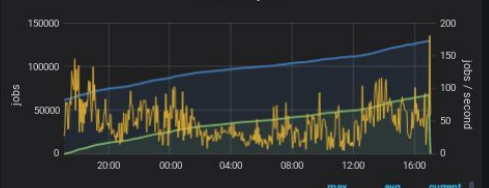
	max	avg	current
number of scheduling cycles	14213	7245	15
scheduling cycles per second (right-y)	21	10	6

processed / pending jobs

	max	avg
average jobs processed in a scheduling cycle	275	268
length of jobs pending queue	9079	4764

Backfill Scheduler

backfilled jobs



	max	avg	current
jobs backfilled since last Slurm start	129780	97439	129780
jobs backfilled since last stats cycle start	67495	35432	267
jobs backfilled per second (right-y)	181	47	60

backfill cycle time

	max	avg	current
last backfilling cycle time	121.8 s	66.3 s	55.5 s
mean backfilling cycle time	111.0 s	55.1 s	111.0 s

last depth cycle

	max	avg	current
processed jobs in last backfill cycle	6689	3120	6689
processed jobs in last backfill cycle (try sched)	2376	1382	2148

backfill queue length

	max	avg
jobs pending backfill processing	9112	4756

Feedback

- Slurm is an amazing piece of software
  - 100,000 jobs/day on heterogeneous system with 1,000s of users and 100s of partitions → 95% utilization
- support is stellar
  - yes, you need support contract with SchedMD)
  - much better than much larger ISVs or HW vendors
  - a few *site down* situations resolved in a matter of hours

Feedback

- but… there **were** *site down* situations
  - a significant number of segfaults
  - major version updates are sometimes a challenge
    - DB conversions, unexpected config changes…
  - x.0 versions not for the faint of heart :)
- bugs.schedmd.com is a great source of information
  - can find a lot of details about Slurm internals there
  - we use it a lot (reported 122 bugs/requests so far)
  - did you know there's a bug report RSS feed?

THANKS!

Any question?

You can find me at **kilian@stanford.edu**

https://github.com/stanford-rc

Stanford
University