





Tuning Slurm the CSCS way

Slurm User Group 2018 Miguel Gila, CSCS September 26, 2018

Three things we do a bit differently

- 1. RM-Replay
- 2. GPU Reporting with Slurm
- 3. Slurm command logging









To be published during SC18

RM-Replay: A High-Fidelity Tuning, Optimization and Exploration Tool for Resource Management

Maxime Martinasso, CSCS

https://github.com/eth-cscs/slurm-replay

RM-Replay

- Like probably every other HPC center out there, we always have (recurrent) users complaining about this:
 - "Why are there available nodes and my jobs are not running??"
- Well, you can't satisfy everyone, but you sure can tune the configuration to be more *effective* in keeping users happier
- Of course, this also depends on users doing something on their side. But that's a different story...





RM-Replay

- How can you evaluate changes in the Slurm configuration and how they affect scheduling and the usage of the machine?
- RM-Replay can replay the submissions done in a period of time and give you an estimation of how busy the machine would have been with the new settings, compared to the original configuration
- Built as a Docker container. Can naturally be executed in Shifter
- With a clever approach it uses unmodified Slurm source code with a few additions to re-play scheduling much faster than real time





How does it work?







How do you use it?



How do we want to use it?

- During development, we've used the tool to identify two important points:
 - Using the switch options increases the fragmentation of the schedule reducing by 10% the job throughput
 - When users provide a better runtime accuracy of their jobs, this decreases the likelihood that their jobs will have a long waiting time in the queue
- Ideally, an auto-tuning framework could potentially make use of this tool in order to automatically configure Slurm and react to change in the job mix
- But for now the plan is to put this on a dedicated system and use it analyze major changes to our configuration and what-if scenarios









GPU Reporting^[*]

[*] Also presented by Nick Cardo at the Cray User Group 2018

Describing the problem

- A batch job is submitted to a compute node containing a GPU
 - Did they utilize the GPU or just the node's processor?
- Easy to tell if a GPU was requested
 - Can check GRES
 - Can check node name
- Hard to tell if a GPU was used from existing accounting
- How to report GPU usage in a meaningful way?





Available tools

nvidia-smi

Nid00032: > nvidia-smi -q -d account	ir	ıg
======NVSMI LOG========	=	
Timestamp	:	Thu May 17 11:52:07
Driver Version	:	384.111
Attached GPUs GPU 00000000:02:00.0	:	1
Accounting Mode	:	Enabled
Accounting Mode Buffer Size	:	1920
Accounted Processes		
Process ID	:	10757
GPU Utilization	:	0 %
Memory Utilization	:	0 %
Max memory usage	:	291 MiB
Time	:	272 ms
Is Running	:	0
Process ID	:	15098
GPU Utilization	:	71 %
Memory Utilization	:	5 %
Max memory usage	:	289 MiB
Time	:	25194 ms
Is Running	:	0
Process ID	:	15125
GPU Utilization	:	93 %
Memory Utilization	:	6 %
Max memory usage	:	289 MiB
Time	:	91777 ms
Is Running	:	0
Process ID	:	4448
GPU Utilization	:	93 %
Memory Utilization	:	6 %
Max memory usage	:	0 MiB
Time	:	91899 ms

RUR

- Tool present only on Cray systems
- Can be used to aggregate data coming from different plugins, including GPU counters
- Needs modifications to be used with native Slurm and not ALPS

Slurm prolog/epilog

 Used to call Cray RUR to start/stop counter collection





How to store data in a meaningful way?

- Store data in Slurm job accounting record
 - Keeps all job data together, no separate database or utilities
 - Reuse an existing text field AdminComment
 - Use JSON format to store multiple pieces of data
- Data is sent to SlurmDBD with a modified RUR plugin that runs at job end

/usr/bin/mysql -h HOST -u DBUSER -pDBPASS DATABASE -e 'update %s_job_table set admin_comment=\"%s\" where id_job=%s and id_user=%s'" % (cluster,jout.replace("\"",'\\"'),jobid,uid)

Extractable with sacct

sacct -o AdminComment

{"gpustats": "maxqpusecs": 146, ← "maxmem": 17034117120,← "summem": 17034117120, "qpusecs": 146 ← Accumlated memory and time } }





Batch Job Summary Report

How to report GPU usage in a meaningful way?







Open questions

- RUR is nice... But perhaps there could be a way to have similar functionality embedded in Slurm itself?
- Would slurmd/slurmctld be able to do such aggregation?
- What about database fields for additional accounting data?
- Jobcomp/ElasticSearch plugin?









Slurm command logging

Tuning Slurm the CSCS way 16

Describing the problem

- **Services** have access to dedicated nodes that query Slurm and/or submit jobs
 - Continuous Integration Systems (Jenkins, etc.)
 - Special frontends (UNICORE, ARC)
- **Users** have access to login nodes to submit jobs
 - Daint has a few login nodes
 - Intended to allow users to submit jobs and build apps
 - But users can basically do whatever they want

So, what do you do when

- User commands start timing out everywhere without any apparent reason?
- Slurmctld logs show it being busy putting tasks on CNs for hours?
 - This somehow tends to happen during weekends or at night...

[2018-09-25T14:41:07.832] debug:	_slurm_rpc_job_pack_alloc_info: JobId=840324 NodeList=nid00007 usec=2	
[2018-09-25T14:41:07.841] debug:	laying out the 1 tasks on 1 hosts nid00007 dist 1	
[2018-09-25T14:41:07.841] debug:	reserved ports 24790 for step 840324.6	
[2018-09-25T14:41:08.592] debug:	freed ports 24790 for step 840324.6	
[2018-09-25T14:41:08.662] debug:	_slurm_rpc_job_pack_alloc_info: JobId=840324 NodeList=nid00007 usec=2	
[2018-09-25T14:41:08.671] debug:	laying out the 1 tasks on 1 hosts nid00007 dist 1	
[2018-09-25T14:41:08.671] debug:	reserved ports 24791 for step 840324.7	



Some lo	ops are evil!
<pre>#!/bin/ba while : do clear squeue squeue sleep 1 done</pre>	grep JOBID grep \${USER}



Jobs with *bajillions* of tasks # sacct -j XXXXXX |wc -1 25337

How do we know what users do?

 Ideally, we would love Slurm to be able to rate-limit the amount of RPCs per user/host/account

- But first, how can we identify precisely what users are doing?
 - Yes, *auditd* is an option...
 - But what's the performance impact of enabling this on Cray's version of the OS?
- What else is out there?





Slurm patch to log user calls

- diff --git a/src/sacctmgr/sacctmgr.c b/src/sacctmgr/sacctmgr.c index ed4ae35c79..1c354dd51a 100644 -- a/src/sacctmgr/sacctmgr.c +++ b/src/sacctmgr/sacctmgr.c @@ -108,6 +108,7 @@ int main(int argc, char **argv) guiet_flag = 0: readonly_flag = 0: verbosity = 0: + log_command_execution_syslog(argc, argv); slurm conf init(NULL); log_init("sacctmgr", opts, SYSLOG_FACILITY_DAEMON, NULL); diff --qit a/src/salloc/salloc.c b/src/salloc/salloc.c index 876c0a8ee5..1303a5f597 100644 --- a/src/salloc/salloc.c
- +++ b/src/salloc/salloc.c

œd -195,6 +195,7 œd int main(int argc, char **argv) slurm_allocation_callbacks_t callbacks; ListIterator iter_req, iter_resp;

+ log_command_execution_syslog(argc, argv);

slurm_conf_init(NULL); debug_flags = slurm_get_debug_flags(); log_init(xbasename(argv[0]), logopt, 0, NULL);

diff --git a/src/common/log.c b/src/common/log.c index 28ace318c4..a755979ec2 100644 --- a/src/common/log.c +++ b/src/common/log.c @@ -79,6 +79,11 @@ #include "src/common/xmalloc.h" #include "src/common/xstring.h" +#include <syslog.h> +#include <stdlib.h> +#include <pwd.h> +#include <libgen.h> #ifndef LINEBUFSIZE # define LINEBUFSIZE 256 #endif <u>@@ -1244,3 +1249,26 @@</u> extern int get_log_level(void) level = MAX(level, log->opt.stderr_level); return level; } +/* Undocumented, CSCS only: logs to syslog the execution of a command */ +void log_command_execution_syslog(int argc, char ** argv){ int i = 1;uid_t uid = geteuid(); struct passwd *pw = getpwuid(uid); static const int BUFFER SIZE = 256; char * buffer = malloc(sizeof(char) * (BUFFER_SIZE + 1)); // if we cannot allocate memory, skip and ignore if (getenv("SLURM_LOG_ACTIONS") && (buffer != NULL)) { for (i=1; i<argc; i++) {</pre> if (strlen(buffer) < BUFFER_SIZE)</pre> strncat(buffer, argv[i], BUFFER_SIZE-strlen(buffer)); else break;

}

setlogmask (LOG_UPTO (LOG_NOTICE));

openlog (basename(argv[0]), LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL1);

```
syslog (LOG_NOTICE, "User: %s, command: %s %s", pw->pw_name, basename(argv[0]), buffer);
closelog ();
```

```
÷.
```

diff --git a/src/common/log.h b/src/common/log.h

index bf55fe10b7..fd429f5761 100644

```
--- a/src/common/log.h
+++ b/src/common/log.h
```

@@ -268,4 +268,6 @@ void debug3(const char *, ...) __attribute__ ((format (printf, 1, 2))); void debug4(const char *, ...) __attribute__ ((format (printf, 1, 2))); void debug5(const char *, ...) __attribute__ ((format (printf, 1, 2)));

+void log_command_execution_syslog(int argc, char ** argv);

Tuning #endif /* !_LOG_H */



















This new information is very useful

- We've detected a few good use cases where we have been able to help users improve their usage of available tools
 - Corner cases where a service needs to query a few hundreds of jobs every few minutes
 - Users that abuse Shell loops or *watch* because they don't know how things work below
 - Usage of scontrol + awesome grep+awk combinations instead of sinfo
 - Insane amounts of parallel sruns, which lead us to adapt GREASY^[*]
- Now we can identify, quickly, when a submission script or a job goes rogue
- Believe it or not, there is so much to learn from users!

^[*] https://user.cscs.ch/tools/high_throughput/





What now?

- Does anybody really need to have squeue open, refreshed every second, 24/7 (even at night) to see if his/her jobs are running??
- Is there any way to rate-limit what users do?
 - We love memcached [*], can it be used here somehow?
- However, this partially highlights that there are valid use cases for alternative ways to access Slurm:
 - RESTful API
 - Fully supported Python/Go bindings
 - PySlurm is really cool, give it a try!

[*] See Nick Cardo's presentation at SLUG17 (https://slurm.schedmd.com/SLUG17/Cardo-SLUG2017.pdf)











Thank you for your attention.