DE LA RECHERCHE À L'INDUSTRIE

Slurm support on large machines

www.cea.fr

SLURM User Group - September 2013 | Matthieu Hautreux <matthieu.hautreux@cea.fr>

Slurm support on large machines
Agenda

Large machine specificities

Methodology and tools

Case studies

Slurm support on large machines

Large machine specificities

Slurm support on large machines
Large machine specificities

Targeted systems

- Petaflopic machines
  - TERA project : Tera-100
    - >4k nodes
    - >130k cores
  - PRACE project : Curie
    - >5k nodes
    - >80k cores

Slurm support on large machines
Large machine specificities

Difficulties

- Uncommon scenarii
  - Approaching the huge figures, you get the unexpected behaviors
- Different user needs, usages or mistakes
  - Different allocation strategies
    - Number of cores per nodes, of cores per sockets, of nodes, exclusive or not
    - Dependencies
  - Exercising a lot of the Slurm (even hidden...) features
  - With different levels of knowledge of what can be done on such systems
    - Exp : srun -n 80000 sinfo ... why is it so slow ?
- Unavailability
  - Slurm as the centerpiece of the resources access
    - Many users impacted, many chances to get a phone call...
    - Many persons waiting for an answer/correction ASAP
- Reproductibility issues
  - Dynamic systems
  - Cost of trying to reproduce a problem at scale is prohibitive

Slurm support on large machines
Large machine specificities

Difficulties

- Distributed architecture
  - A lot of components bound by complex networks
  - The problem can be anywhere and anywhere is large

- A needle in the hay effect...
  - Large volume of logs to analyze
    - One single slurmdbd log file, ~ok
    - One single slurmctld log file, ~ok
    - One slurmd log file per compute node, ko...
      - But can be centralized using a distributed syslog architecture
  - Large amount of nodes to analyze
    - In terms of process states
    - In terms of communication states
  - Large volume of RPCs to analyze on the controller

Slurm support on large machines

Methodology and tools

Slurm support on large machines
Methodology and tools

Methodology

- Be ready for the unexpected
  - $ head -n 1 /etc/sysconfig/slurm
    ulimit -c unlimited
  - Check/validate core file generation at installation time
    - « scontrol abort »
  - Automate the restart of the controller
    - « service slurm status || service slurm restart » in crontab (or equivalent)
      - Unavailability should be reduced to less than the comm timeout
    - Help to let the system available as much as possible
      - Autonomous job submissions
  - Ensure that you have the corresponding sources available

- When the unexpected comes...
  - Identify the consequences of the issue
    - Fatal : bugs, assertions or slurm components aborts
    - Missing feature(s) : no more submission, no more fairsharing, no more backfilling, no more check_node_health, ...
    - Missing resources : large number of {completing/unresponding/drain/...} nodes
    - Job related issues : jobs failing to start/stop, steps no longer created, ...
    - Performances : slow/unresponsive system, slowdowns trigerring wdg/timeouts, ...

Methodology

- When the unexpected comes... (continued)
  - Try to define the issue family
    - Comm issues, DOS-like issues, Deadlock, Data corruption, Race condition, ...
  - Try to identify the origin of the issue
    - Collect all the relevant traces as long as you can (and think after)
      - Issues can be transient and disappear as they appear
      - Get relevant slurm log sections of every components
      - Get « live » core dump using gcore
        > save the associated binary too
        > get it twice in case of possible deadlocks (easier to see what is frozen)
      - Get processes states and open files
      - Get systems states if possible
      - Get communications states too (TCP sockets states)
      - Store all of that in a directory and archive it
        > you could need it in 6 or 12 month...

Methodology

- When the unexpected comes... (continued)
  - Try to make things work again ASAP (remember the unavailability drawback)
    - Try to identify the faulty user/node/job
      - If you are lucky, that will resolve the immediate issue, but will still require investigation to avoid future occurences
      - Commonly the case for DOS-like and Comm issues
    - Restart the faulty components
      - It might work ... but will certainly destroy most of the evidences/clues
      - Commonly the case for Deadlocks and Race conditions issues
    - Bypass the faulty subcomponent temporarily if necessary and possible
      - Commonly the case for Data corruption issues (might be others too)
    - If it still does not work...
      - Commonly the case for Data corruption issues

        > First backup your controller state files and jobs information (you might loose them)
      - You 're entering a live session, you should get spectators sooner or later...

Methodology

- Once worked around (or in live session in the worst cases)
  - Look at the logs
    - grep&sed[[&grep&sed]...] (in parallel on all the compute nodes if necessary)
    - It could provide useful error messages
      - Easily searchable in the source code
  - Look at the traces
    - From core files to identify the suspicious calls
      - Compare with a valid behavior to see the differences
      - A lot of data are global in Slurm so you can easily access the vars using gdb
      - Easily searchable in the source
    - From open files / sockets stats to identify the communicators and their states
  - Look at the source
    - For the suspicious calls and use « git blame » to understand the reasons of the logic
  - Make and then test your patches
    - or ask for support on the mailing list with all the collected material ready :-)

Useful tools or files on nodes

- Systems inspection
  - /var/log/slurm*, /var/log/*
  - ps, top, vmstat
  - perf, oprofile
  - lsof, nodeset *(from clustershell package)*
- Processes inspection
  - gdb, gstack, gcore, crash
  - /proc/$pid/*
  - lsof, nodeset *(from clustershell package)*
  - strace, ltrace
- Comm inspections
  - netstat, lsof, ss
  - tcpdump *(iptables to reproduce packet loss)*
- Sources inspection
  - emacs, cscope-mode (or grep:), vi / cscope, or the equivalent ...)
  - git (blame)
  - Man pages
  - Slurm mailing-list archive, SchedMD Bugzilla

Aggregation of results

- Try to aggregate information when possible
  - **Clustershell** (pdsh/dshback replacement) to parallelize the processing of nodes traces and aggregate results
    - Need to perform the right « clush ... \| sed 's/xxxxx/xxxxx/' ...» to anonymize the outputs and get the general trend to discriminate the most suspicious nodes
    - Slurm would gain in having stronger log format conventions
  - **clustack** to acquire aggregated information of gstack outputs
    - In-house dev based on clustershell python API
    - Useful to read multithreaded apps gstack outputs with large outputs
      - With hundreds of threads for example...
    - Parallel execution support to aggregate among multiple nodes

Slurm support on large machines
Methodology and tools

Aggregation of results

- clustack example

```
[root@inti0 ~] # clustack -bar pgrep:slurmctld
Thread 1:
#3 0x..... in main ()
#2 0x..... in _slurmctld_background ()
#1 0x..... in sleep () from /lib64/libc.so.6
#0 0x..... in nanosleep () from /lib64/libc.so.6
...
Thread 6:
#4 0x..... in clone () from /lib64/libc.so.6
#3 0x..... in start_thread () from /lib64/libpthread.so.0
#2 0x..... in _decay_thread () from /usr/lib64/slurm/priority_multifac
#1 0x..... in sleep () from /lib64/libc.so.6
#0 0x..... in nanosleep () from /lib64/libc.so.6
...
[root@inti0 ~] #
```

```
[hautreux@inti50 ~]$ srun -n 700 sinfo >/dev/null
[hautreux@inti50 ~]$
```

```
[root@inti0 ~] # clustack -bar pgrep:slurmctld
Thread [2-37,39-84,87-123,125-137,139-153,155,165-175]:
#6 0x..... in clone () from /lib64/libc.so.6
#5 0x..... in start_thread () from /lib64/libpthread.so.0
#4 0x..... in _service_connection ()
#3 0x..... in slurm_receive_msg ()
#2 0x..... in _slurm_msg_recvfrom_timeout ()
#1 0x..... in _slurm_recv_timeout ()
#0 0x..... in poll () from /lib64/libc.so.6
Thread [38,85-86,124,138,154]:
#8 0x..... in clone () from /lib64/libc.so.6
#7 0x..... in start_thread () from /lib64/libpthread.so.0
#6 0x..... in _service_connection ()
#5 0x..... in slurmctld_req ()
#4 0x..... in _slurm_rpc_dump_nodes ()
#3 0x..... in slurm_send_node_msg ()
#2 0x..... in _slurm_msg_sendto_timeout ()
#1 0x..... in _slurm_send_timeout ()
#0 0x..... in poll () from /lib64/libc.so.6
Thread 1:
#3 0x..... in main ()
#2 0x..... in _slurmctld_background ()
#1 0x..... in sleep () from /lib64/libc.so.6
#0 0x..... in nanosleep () from /lib64/libc.so.6
...
Thread 160:
#4 0x..... in clone () from /lib64/libc.so.6
#3 0x..... in start_thread () from /lib64/libpthread.so.0
#2 0x..... in _decay_thread () from /usr/lib64/slurm/priority_multifactor.so
#1 0x..... in sleep () from /lib64/libc.so.6
#0 0x..... in nanosleep () from /lib64/libc.so.6
...
[root@inti0 ~] #
```

Debugging large machines

Case study 1

Slurm support on large machines
Case study 1

Fairshare logic is blocked

- October 28th, 2013, no more evolution of the fairshare priorities and account usages
  - Everything else sounds correct
  - Seems to be a « missing feature scenario »
  - Get core files using « gcore $(pgrep slurmctld) »
    - Should be sufficient for such a class of problem
  - Restart the controller, everything is good again, great !
  - Start to look at the traces
    - The decay thread seems to work but is sleeping
    - Look at the code and check the variables...

In priority_multifactor.c:1075

```
/* sleep for calc_period secs */
tm.tm_sec += calc_period;
tm.tm_isdst = -1;
next_time = mktime(&tm);
sleep((next_time-start_time));
start_time = next_time;
```

man 3 sleep :

SYNOPSIS

#include <unistd.h>

unsigned int sleep(unsigned int seconds);

```
(gdb) t 5
[Switching to thread 5 (Thread 0x2b996c202700 (LWP 8339))]#0 0x00000031bc0ab15d in nanosleep () from /lib64/libc.so.6
(gdb) bt
#0 0x00000031bc0ab15d in nanosleep () from /lib64/libc.so.6
0000001 0x00000031bc0aafd0 in sleep () from /lib64/libc.so.6
0000002 0x00002b9898307486 in _decay_thread (no_data=<value optimized out>) at priority_multifactor.c:1075
0000003 0x00000031bcc077f1 in start_thread () from /lib64/libpthread.so.0
0000004 0x00000031bc0e5ccd in clone () from /lib64/libc.so.6
(gdb) select 2
(gdb) print start_time
$2 = 1351386075
(gdb) print (unsigned int) next_time - start_time
$1 = 4294963996
(gdb)
```

Slurm support on large machines
Case study 1

Fairshare logic is blocked

- 4294963996 seconds, quite a long sleep ! (>130 years)
  - We just need to exercise more patience, it would eventually work :-)
- 1351386075 (start_time) is an interesting date (in epoch...)
  - date -u -d @1351386075
    Sun Oct 28 01:01:15 UTC 2012
  - This corresponds to the shift time for daylight saving time
- The good news is that we have a year to find the bug...
  - So we ask for support ...
  - And the bug is known and corrected !!!
    Bugzilla 175, commit 1d90cd35ff6621717911edc752af014a32360934

Debugging large machines

Case study 2

Slurm support on large machines
Case study 2

Slurmctld periodically unresponsive

- Users complain about periodic failure of their submissions
  - Admins confirm that even sinfo-like commands time out
- Seems to be a performance issue
  - Collect slurmctld info for that
    - **top** confirms the high load of the process
    - **clustack** shows a high number of threads processing rpc_dump_nodes
    - **lsof** provides the name of the nodes involved in RPCs
    - Looking at the jobs versus the involved nodes, a user is highlighted
    - Looking at the user's app behavior, a bad usage of slurm is found
      - A large number of concurrent squeue calls to get local task information (one per task, thousands of tasks, several sequential steps per job...)
      - A « distributed denial of service » !!!
    - The user is contacted and the app corrected
- Just the consequences of doing a small mistake at scale...

Slurm support on large machines
Case study 2

Slurmctld periodically unresponsive

- Details

```
$ top
64829 slurm     20   0 16.4g 108m 6124 S 166.0  0.2   4205:24 slurmctld
$ clustack
...
 ---------------
 Thread[2-46,48-71,73-190,192-216,218-249,259-265] (251)
 ---------------
 0x..... in clone () from /lib64/libc.so.6
 0x..... in start_thread () from /lib64/libpthread.so.0
 0x..... in _service_connection ()
 0x..... in slurmctld_req ()
 0x..... in _slurm_rpc_dump_nodes ()
 0x..... in lock_slurmctld ()
 0x..... in _wr_wrlock ()
 0x..... in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
...
$ lsof -p 64829 | grep ESTABLISHED | sed 's/.*\->\([a-z0-9]*\)\..*/\1/' | nodeset -f
curie[0,71,1825-1827,1831,1837,1872-
1873,1938,2280,2994,2996,2998,3002,3304,3312,3506,3721,3931,3945,4293,4299,4307,4508,4522,4560,4562,4568,4574,4653,4657,4663,4665,5118,
5121-5123,5212,5218,5222,5224,5288,5391,5395,5397,5399,5403,5554,5558,5561,5563,5565,5642,5645-5646,5651,5655,5932,5934,5936,6402,6408]
$ squeue -w curie[....]
...
$ ssh ...
```

Debugging large machines

Case study 3

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

- Users/admins complains about large MPI jobs finishing correctly but staying allocated until their time limit
  - Large waste of resources for users not specifying correcly their execution time
- Seems to be a deadlock issue at first
  - But not so easy !
  - Only the srun command is present while the job is blocked !
    - All the slurmstepds have already finished their execution
  - Seems to be a deadlock-like scenario in distributed environment
    - so maybe a comm issue too...
- We have a systematic reproducer !!! it only requires 4700 nodes ....
  - Need to do step by step analysis when debugging slots are allowed....
    - 1 hour every 2 months during maintenance period, great :-)
  - It shows that srun hangs waiting for stdout,err messages from a few hundreds nodes
    - But not necessary the same distribution of nodes over the different runs...

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

- Interesting additional inputs !
  - The issue only appears when the OpenMPI app is launched directly using srun –resv-port
    - It does not appear when salloc/mpirun is used instead....
- It's time to get information from the system, we only have one hour....
  - over the 4700+1 (login) nodes involved
    - Start a tcpdump collection
    - Start a « ss » periodic collection
  - Launch the app and wait for its workload termination
  - As soon as it finishes
    - Get core files from the hung srun process(es)
    - Get its lsof result and identify the involved failed nodes
  - Connect to one of the failed nodes for live analysis
    - Look at network live statistics to see what happens

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

The srun stack :

```
[root@curie52 ~] # gstack 37967
Thread 3 (Thread 0x2b30373cb700 (LWP 37969)):
#0  0x00000035b7a0f2a5 in sigwait () from /lib64/libpthread.so.0
#1  0x0000000004237ac in _srun_signal_mgr ()
#2  0x00000035b7a07851 in start_thread () from /lib64/libpthread.so.0
#3  0x00000035b6ee767d in clone () from /lib64/libc.so.6
Thread 2 (Thread 0x2b30375cd700 (LWP 37972)):
#0  0x00000035b6eddfc3 in poll () from /lib64/libc.so.6
#1  0x000000000042b605 in eio_handle_mainloop ()
#2  0x0000000004e81a2 in _io_thr_internal ()
#3  0x00000035b7a07851 in start_thread () from /lib64/libpthread.so.0
#4  0x00000035b6ee767d in clone () from /lib64/libc.so.6
Thread 1 (Thread 0x2b3031d88040 (LWP 37967)):
#0  0x00000035b7a080ad in pthread_join () from /lib64/libpthread.so.0
#1  0x0000000004e7f6d in client_io_handler_finish ()
#2  0x0000000004e8fc2 in slurm_step_launch_wait_finish ()
#3  0x0000000004222c2 in srun ()
#4  0x00000035b6e1ecdd in __libc_start_main () from /lib64/libc.so.6
#5  0x0000000000421129 in _start ()
[root@curie52 ~] #
```

The involved compute nodes :

```
[myuser@curie52 heavyp2p]$ lsof -p 37967 |  grep ESTABLISHED | sed 's/.*\->\([a-z0-9]*\)\..*/\1/' | nodeset -f
Curie[1675,1680,1843,1862,1871,1882,1891,1898,....,5793,.....,6245,6273,6367]
[myuser@curie52 heavyp2p]$ lsof -p 37967 |  grep ESTABLISHED | sed 's/.*\->\([a-z0-9]*\)\..*/\1/' | nodeset -c
220
[myuser@curie52 heavyp2p]$
```

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

On one of the involved node, a large number of closing/half-closed sockets :

```
[myuser@curie5793 ~]$ ss | awk '{print $1}' | sort | uniq -c
   1153 CLOSING
      6 ESTAB
    675 FIN-WAIT-1
    152 LAST-ACK
      1 State
[myuser@curie5793 ~]$ cat /proc/net/sockstat
sockets: used 475
TCP: inuse 1996 orphan 1980 tw 416 alloc 2001 mem 1309
UDP: inuse 14 mem 5
[myuser@curie5793 ~]$
```

After a while, all the orphan sockets are purged by the kernel :

```
[myuser@curie5793 ~]$ cat /proc/net/sockstat
sockets: used 476
TCP: inuse 17 orphan 0 tw 0 alloc 22 mem 1
UDP: inuse 14 mem 5
[myuser@curie5793 ~]$
```

The orphan sockets are mostly due to the OpenMPI runtime, when used in direct slurm execution, that opens a large number of TCP sockets in the case of the heavyp2p benchmark and lets the kernel close them at exit :

```
[myuser@curie5793 ~]$ ss | awk '{print $NF}' | sed 's/[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*/x\.x\.x\.x\./g' | sort | uniq -c | sort -n  | tail -n 1
   1693 x.x.x.x.:14304
[myuser@curie5793 ~]$ ss | awk '{print $(NF-1)}' | sed 's/[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*/x\.x\.x\.x\./g' | sort | uniq -c | sort -n  | tail -n 1
    286 x.x.x.x.:14304
[myuser@curie5793 ~]$ grep MpiParams /etc/slurm/slurm.conf
MpiParams=ports=12000-16999
[myuser@curie5793 ~]$
```

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

But slurmstepd also lets the kernel manage the termination of IO redirection
-> so IO redirection sockets compete in the kernel with MPI apps sockets after processes exits.

Orphan sockets are managed in a special manner in the kernel :

```
[root@curie52 ~] #  cat /proc/sys/net/ipv4/tcp_orphan_retries
0
[root@curie52 ~] #
```

-> 0 means that you'll have 8 retries exponentially spread around 50s
-> sockets closed due to processes exits at the same time will have timely adjusted retries

-> we may have periodic burst of retries resulting in packet losses...

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

Looking at the tcpdump outputs, we see that this is the right direction :

> On the srun node, we can see that we do not receive the message 3147758607:3147758617 and ask for its retransmission

15:39:05.599940 IP (tos 0x0, ttl 62, id 24308, offset 0, flags [DF], proto TCP (6), length 60)
curie5793.57268 > **curie52**.55731: Flags [S], cksum 0x84d3 (correct), seq **3147758580**, win 14600, options [...], length 0
15:39:05.600052 IP (tos 0x0, ttl 64, id 6170, offset 0, flags [none], proto TCP (6), length 60)
curie52.55731 > **curie5793**.57268: Flags [S.], cksum 0x211a (correct), seq 2091871385, **ack 3147758581**, win 65160, ..., length 0
15:39:05.600329 IP (tos 0x0, ttl 62, id 24309, offset 0, flags [DF], proto TCP (6), length 52)
curie5793.57268 > **curie52**.55731: Flags [.], cksum 0x4e00 (correct), **seq 3147758581**, ack 2091871386, win 115, options [...], length 0
15:39:05.600440 IP (tos 0x0, ttl 62, id 24310, offset 0, flags [DF], proto TCP (6), length 78)
curie5793.57268 > **curie52**.55731: Flags [P.], cksum 0x6b71 (correct), **seq 3147758581:3147758607**, ack 2091871386, ..., length 26
15:39:05.600455 IP (tos 0x0, ttl 64, id 6171, offset 0, flags [none], proto TCP (6), length 52)
curie52.55731 > **curie5793**.57268: Flags [.], cksum 0x4e39 (correct), **seq 2091871386**, **ack 3147758607**, win 32, options [...], length 0
15:44:39.147855 IP (tos 0x0, ttl 62, id 24317, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > **curie52**.55731: Flags [FP.], cksum 0xbb68 (correct), **seq 3147758617:3147758627**, ack 2091871386, ..., length 10
15:44:39.147869 IP (tos 0x0, ttl 64, id 6178, offset 0, flags [none], proto TCP (6), length 64)
curie52.55731 > **curie5793**.57268: Flags [.], cksum 0x265a (correct), **seq 2091871386**, **ack 3147758607**, win 32, options [...], length 0

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

Looking at the tcpdump outputs, we see that this is the right direction : (continued...)

> On the compute node, we can see that we sent multiple times the 3147758607:3147758617 segment, even receive the retransmission request, but never succeed in sending it until the discard of the socket by the kernel

15:44:30.952740 IP (tos 0x0, ttl 64, id 24311, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:31.157547 IP (tos 0x0, ttl 64, id 24312, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:31.577216 IP (tos 0x0, ttl 64, id 24313, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:32.417194 IP (tos 0x0, ttl 64, id 24314, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:34.097238 IP (tos 0x0, ttl 64, id 24315, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:37.457210 IP (tos 0x0, ttl 64, id 24316, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:39.147555 IP (tos 0x0, ttl 64, id 24317, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [FP.], cksum ..., **seq 3147758617:3147758627**, **ack 2091871386**, win 115, ..., length 10
15:44:39.147651 IP (tos 0x0, ttl 62, id 6178, offset 0, flags [none], proto TCP (6), length 64)
curie52.55731 > **curie5793**.57268: Flags [.], cksum ..., **seq 2091871386**, **ack 3147758607**, win 32, ..., length 0
15:44:44.177196 IP (tos 0x0, ttl 64, id 24318, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:44:57.617207 IP (tos 0x0, ttl 64, id 24319, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10
15:45:24.497212 IP (tos 0x0, ttl 64, id 24320, offset 0, flags [DF], proto TCP (6), length 62)
curie5793.57268 > curie52.55731: Flags [P.], cksum ..., **seq 3147758607:3147758617**, **ack 2091871386**, win 115, ..., length 10

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

- Conclusions...
  - The problem only appears with « srun --resv-ports »
    - this mode seems to let the OpenMPI runtime create a lot of TCP sockets for processes having a large numbers of connected peers without closing them properly
  - Slurmstepd does not close properly its IO redirection sockets either
  - OpenMPI apps and slurmstepd(s) then compete during the bursts of orphan sockets closures
    - Not a big deal for the MPI job, it is already finished
    - But that may block the srun and lead to our issue

Slurm support on large machines
Case study 3

Large jobs hang after end of execution until time limit

- So ?
  - SchedMD Bugzilla #149
  - Work-around by SchedMD adding a timeout in srun to abort the IO redirection instead of waiting indefinitely
  - Keepalive addition by SchedMD in the comm logic to avoid this kind of issues in other components exchanges in the future
  - Thoughts about refactoring the IO-redirection protocol to properly manage the sockets closures and not be worried by competing orphan sockets
    - Implementing a shutdown logic in the exchanges

Thank you for your attention

Questions ?

Commissariat à l'énergie atomique et aux énergies alternatives
Centre DAM Ile-de-France | Bruyères-le-Châtel 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 |
Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019

DAM/DIF
DSSI
SISR