burst_buffer/lua and slurmscriptd

Marshall Garey
SchedMD

Slurm User Group Meeting 2021

Agenda

All times are US Mountain Daylight (UTC-6)

Time	Speaker	Title
9:00 - 9:50	Jason Booth	Field Notes 5: From The Frontlines of Slurm Support
10:00 - 10:25	Nate Rini	REST API *and also* Containers
10:30 - 10:50	Marshall Garey	burst_buffer/lua and slurmscriptd
11:00 - 11:25	Nick Ihli	Slurm in the Clouds
11:30 - 11:50	Tim Wickberg	Slurm 21.08 and Beyond

Welcome

- Five separate presentations, five separate streams
- Presentations will remain available for at least two weeks after SLUG'21 concludes
- Presentations are available through the SchedMD Slurm YouTube channel
  - https://youtube.com/c/schedmdslurm
- Or through direct links from the agenda
  - https://slurm.schedmd.com/slurm_ug_agenda.html

Asking questions

- Feel free to ask questions throughout through YouTube's chat
- Chat is moderated by SchedMD staff
  - Tim McMullan, Ben Roberts, and Tim Wickberg
  - Also identified by the little wrench symbol next to their name
- Questions will be relayed to the presenter by the moderators
  - Some may be deferred to the end if they cannot be relayed in a timely fashion
  - Or some may be answered by the moderators in chat directly

burst_buffer/lua and slurmscriptd

Marshall Garey
SchedMD

burst_buffer/lua

Introduction to burst buffer

- Burst buffers are intermediate storage between the slow long-term storage and the compute nodes
- A way to move data from slow storage to faster storage (closer to the compute node) while a job is pending
- Increase job performance
- Avoid moving data during valuable compute time

Slurm burst_buffer/datawarp plugin

- slurmctld executes Cray's datawarp script at different points in a job's lifecycle:
  - Job submission
  - Job pending
  - Job allocated, not running yet
  - Job finished running, in completing state
  - Job totally complete
- Slurm does **not** stage data; Cray's datawarp script does the hard work

New burst_buffer/lua plugin

- Motivation: provide a generic burst buffer plugin
- Similar to datawarp plugin
  - Provide hooks to a script named "burst_buffer.lua" written by a system administrator
  - Call burst_buffer.lua at specific points in a job's life cycle
- The script can do **anything**
  - "Burst buffer" is a misnomer
  - Asynchronous calls to a generic script is the interesting part

When is burst_buffer.lua called?

Job state	**burst_buffer.lua function**
Job submission	slurm_bb_job_process
Job pending	slurm_bb_data_in
Job allocated resources, not running yet	slurm_bb_pre_run
Job finished, completing state	slurm_bb_data_out
Job completed	slurm_bb_teardown

Potential uses for burst_buffer/lua

- Move data from slow storage to faster storage closer to the compute nodes
- Move data from on-site storage to the cloud
    - Don't pay for the cloud nodes until the data is in the cloud
- Anything that the system administrator wants

burst_buffer/lua - How to use

- Job requests using burst_buffer/lua
  - Job script contains "#BB_LUA"

```
!/bin/bash
#SBATCH -t 1
#SBATCH -q speedy
#BB_LUA
script
script
```

burst_buffer/lua - How to use

- Job requests using burst_buffer/lua
    - Or command-line option
        - `$ sbatch --bb=”#BB_LUA”`
    - System administrator defines and enforces additional syntax with burst_buffer.lua
- System administrator implements burst_buffer.lua
    - Template provided with Slurm: burst_buffer.lua.example

burst_buffer.lua - Example

```
$ cat bb_example.batch
#BB_LUA stage_in source="/home/marshall/hello.txt" destination="/tmp/job_in"
#BB_LUA stage_out source="/tmp/job_out" destination="/home/marshall/job_out"
echo "my job $SLURM_JOB_ID output this" > /tmp/job_out

$ sbatch -Dtmp burstbufferjobs/bb_example.batch
Submitted batch job 170270
$ cat /tmp/job_in /tmp/job_out job_out
hello
my job 170270 output this
my job 170270 output this
```

burst_buffer.lua - Example

```
function slurm_bb_job_process(job_script)
    -- Variables are initialized to nil if no value given
    local rc
    -- Discard all return values except rc
    rc = _parse_job_script(job_script, true, true)
    if (rc == slurm.ERROR) then
      return slurm.ERROR, "Burst buffer staging directive not given or invalid"
    end
    return slurm.SUCCESS
end
```

burst_buffer.lua - Example

```
function slurm_bb_data_in(job_id, job_script)
    return _stage(job_script, "stage_in")
end

function slurm_bb_data_out(job_id, job_script)
    return _stage(job_script, "stage_out")
end
```

burst_buffer.lua - Example

```
function _stage(job_script, stage)
    local rc, src, dest, tmp1, tmp2
    if (stage == "stage_in") then
      rc, src, dest, tmp1, tmp2 = _parse_job_script(job_script, true, false)
    else -- assume stage_out
      rc, tmp1, tmp2, src, dest = _parse_job_script(job_script, false, true)
    end
    if (rc == slurm.ERROR) then
      return slurm.SUCCESS -- Job doesn't want this stage, return success
    end
    return _stage_file(src, dest)
end
```

burst_buffer.lua - Example

```
function _parse_job_script(job_script, stage_in, stage_out)
    local rc, in_src, in_dest, out_src, out_dest
    local rc1, rc2 = slurm.SUCCESS, slurm.SUCCESS

    io.input(job_script)
    local contents = io.read("*all")
    io.close()
    if (stage_in) then
      rc1, in_src, in_dest = _get_stage_src_dest(contents, "#BB_LUA stage_in")
    end
    if (stage_out) then
      rc2, out_src, out_dest = _get_stage_src_dest(contents, "#BB_LUA stage_out")
    end
```

burst_buffer.lua - Example

```
function _parse_job_script(job_script, stage_in, stage_out)
    -- _parse_job_script continued
    if ((rc1 == slurm.ERROR) and (rc2 == slurm.ERROR)) then
      rc = slurm.ERROR
    else
      rc = slurm.SUCCESS
    end
    return rc, in_src, in_dest, out_src, out_dest
end
```

burst_buffer.lua - Example

```lua
function _get_stage_src_dest(contents, stage_str)
    local src, dest, inx_start, inx_end
    -- Lazy regex; require path between quotes and do not allow spaces
    inx_start, inx_end, src, dest = string.find(contents, stage_str .. " source=(\"%S+\")
destination=(\"%S+\")")
    if (inx_start == nil) then
      return slurm.ERROR, nil, nil
    end
    slurm.log_info("stage=%s: src=%s, dest=%s", stage_str, src, dest)
    return slurm.SUCCESS, src, dest
end
```

burst_buffer.lua - Example

```
function _stage_file(src, dest)
    local rc, str, num, ret_str
    if ((src == nil) or (dest == nil)) then
      return slurm.ERROR, "src or dest are nil"
    end
    cmd = "cp " .. src .. " " .. dest
    slurm.log_info("Running %s", cmd)
    rc, str, num = os.execute(cmd)
    if (rc == nil) then
      ret_str = cmd .. " failed; " .. str .. ":" .. num
      return slurm.ERROR, ret_str
    end
    return slurm.SUCCESS
end
```

burst_buffer.lua called in two ways

- Some functions are called synchronously
  - Called directly from slurmctld is faster than fork()'ing a new process
  - ⚠ Cannot be killed, must run quickly! ⚠
  - Example: slurm_bb_job_process - called on job submission while slurmctld holds locks
- Some functions are called asynchronously
  - Slurm fork()'s a new process, then calls burst_buffer.lua
  - Can be killed (timeout, job cancel, slurmctld shutdown)
  - Examples: slurm_bb_data_in, slurm_bb_data_out

burst_buffer/lua - Documentation

- https://slurm.schedmd.com/burst_buffer.html
- https://slurm.schedmd.com/burst_buffer.conf.html
- https://slurm.schedmd.com/slurm.conf.html#OPT_BurstBufferType

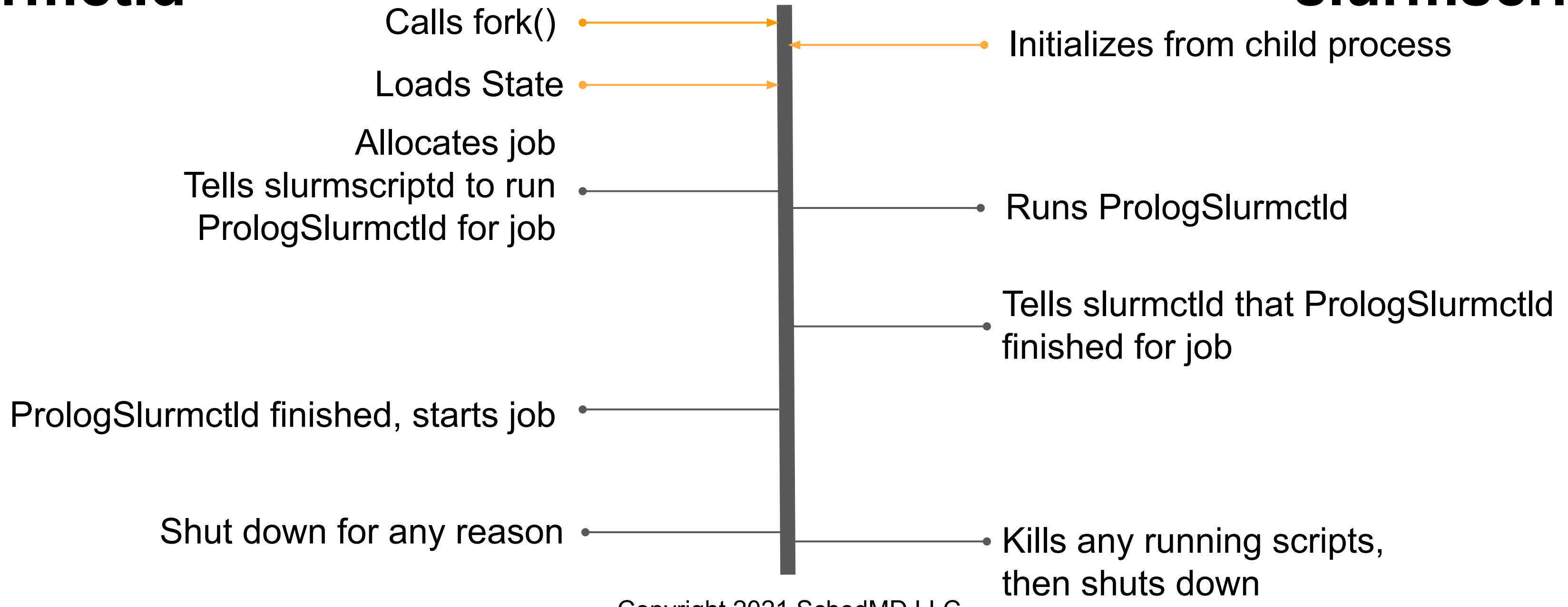
slurmscriptd

Problem: fork() is slow

⚠️ Current functionality and Problem ⚠️

- slurmctld calls fork()/exec() to run many scripts
  - Examples: {Prolog,Epilog}Slurmctld, burst_buffer/datawarp
- Test with PrologSlurmctld and EpilogSlurmctld:
  - sbatch --array=1-10000 --wrap='srun hostname'
  - Time from earliest submit time to last end time
  - slurmctld with minimal memory footprint: **4 min 56 sec**
  - slurmctld with 5 GB memory footprint: **18 min 5 sec**
- Solution: use a different "daemon" to run scripts

How slurmscriptd works - startup



slurmctld

Calls fork()

Loads State

Allocates job
Tells slurmscriptd to run
PrologSlurmctld for job

PrologSlurmctld finished, starts job

Shut down for any reason

slurmscriptd

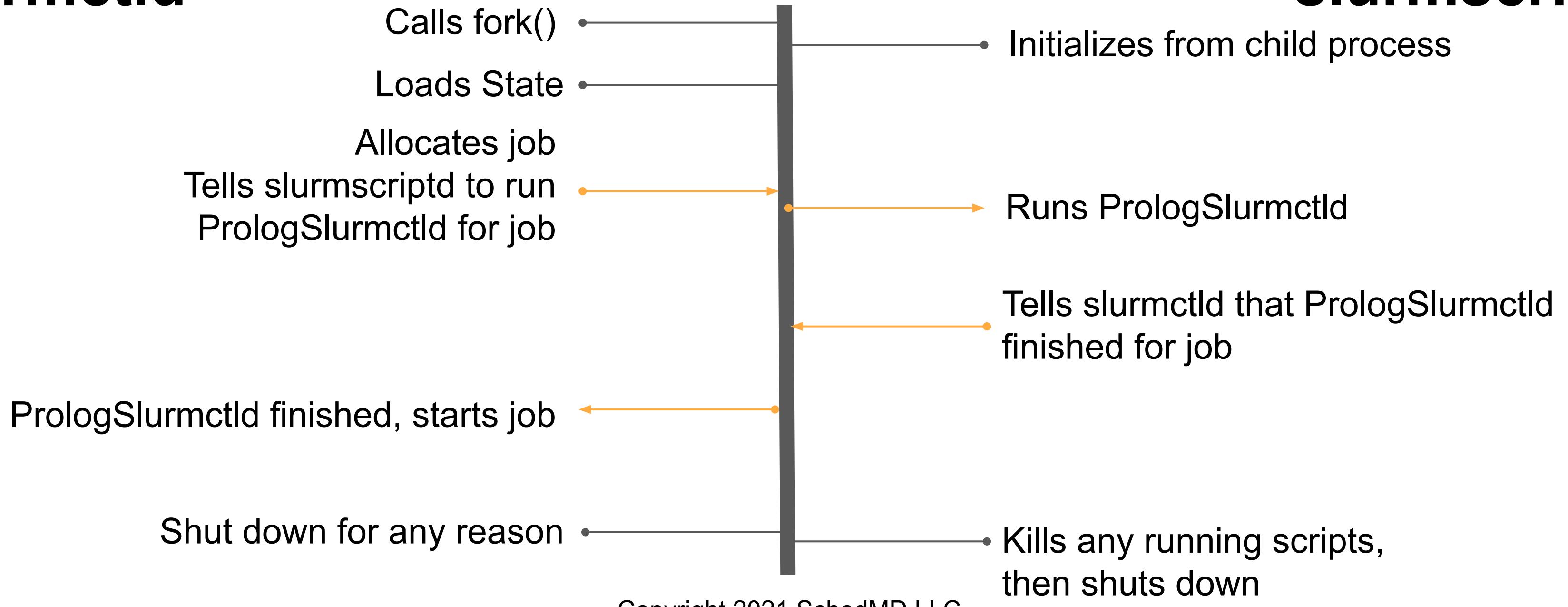
Initializes from child process

Runs PrologSlurmctld

Tells slurmctld that PrologSlurmctld
finished for job

Kills any running scripts,
then shuts down

How slurmscriptd works - run scripts



slurmctld

Calls fork()

Loads State

Allocates job
Tells slurmscriptd to run
PrologSlurmctld for job

PrologSlurmctld finished, starts job

Shut down for any reason

slurmscriptd

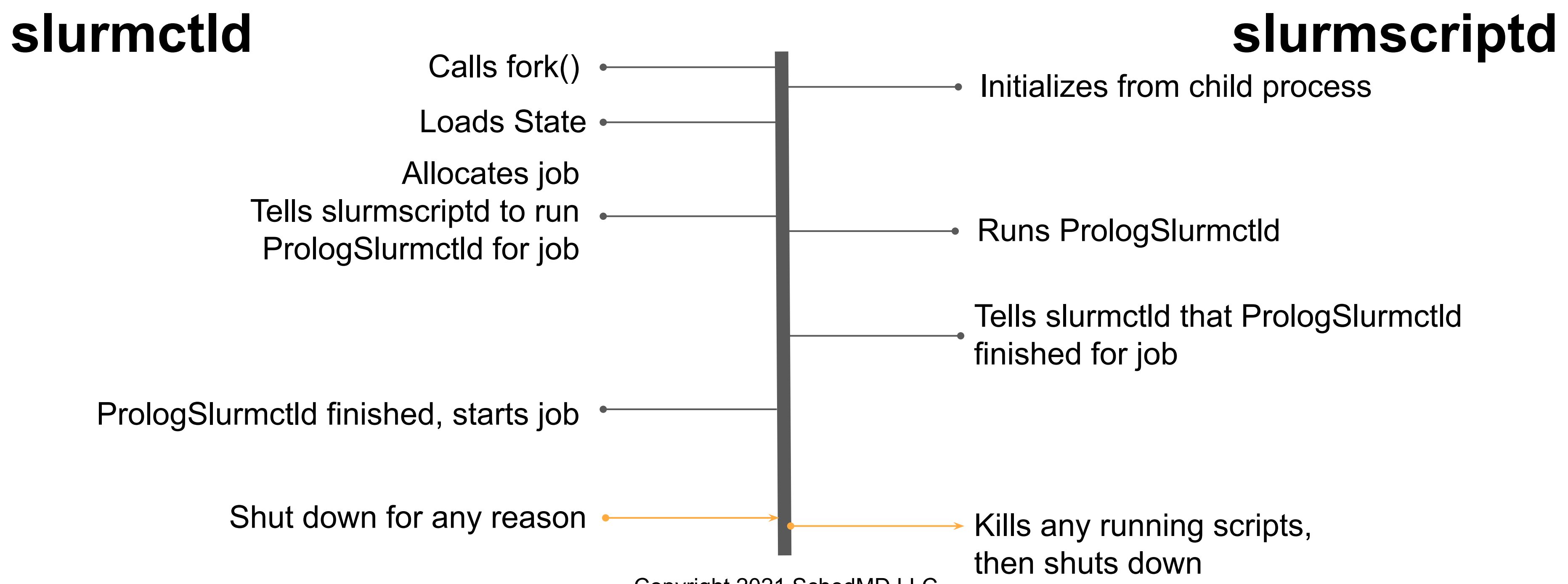
Initializes from child process

Runs PrologSlurmctld

Tells slurmctld that PrologSlurmctld
finished for job

Kills any running scripts,
then shuts down

How slurmscriptd works - shutdown



slurmscriptd Improves Performance

- Run 10,000 jobs with PrologSlurmctld and EpilogSlurmctld
- slurmctld with 5 GB memory footprint
  - Slurm <= 20.11 (without slurmscriptd): **18 min 5 sec**
  - Slurm 21.08 (with slurmscriptd): **4 min 23 sec**

slurmscriptd

- Currently runs PrologSlurmctld, EpilogSlurmctld, and asynchronous calls to burst_buffer.lua
- Potential future work: remove most or all fork() calls from slurmctld
  - MailProg
  - SuspendProgram/ResumeProgram
  - strigger programs

Questions?

Next Session

- The next presentation is by Nick Ihli: "Slurm in the Clouds"
- Starts at 11am Mountain Daylight Time (UTC-6)
- And is on a separate YouTube Live stream
- Please see the SchedMD Slurm YouTube channel for links

End Of Stream

- Thanks for watching!