Step Management Enhancements

Brian Christiansen
What is a job and a step?

- **Job**
  - An allocation of resources (nodes, cpus, memory, gpus, licenses, etc.)

```bash
#!/bin/bash
#SBATCH -n6
...
```
What is a job and a step?

- **Step**
  - Allocated resources of the job that run processes/tasks
  - Can carve up allocated resources into multiple steps
  - Steps typically run parallel programs (e.g. MPI)
  - Steps can run serially or concurrently

```
#!/bin/bash
#SBATCH -n6
srun -n4 --exact a.out &
srun -n2 --exact b.out &
wait
```
What is a job and a step?

- **batch step**
  - Runs the batch script

- **interactive step**
  - Step created on compute node with a PTY for interactive work inside allocation

- **extern step**
  - Enabled through PrologFlags=contain
  - Runs on all slurmds
  - Used to:
    - Launch prologs before allocation
    - Setup X11 forwarding, containers
    - Track processes started outside of allocation (e.g. pam_slurm_adopt, ssh, mpi)
    - Did you know? mpirun typically uses srunk to launch the task

- **job step**
  - execs and shepherds processes
Step Launch Sequence

1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
Step Launch Sequence

1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
Step Launch Sequence

1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
Step Launch Sequence

1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O and launches tasks
Step Launch Sequence

1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O and launches tasks
8. On task termination, `slurmstepd` notifies `srun`
9. `slurmstepd` sends step completions to `slurmctld` (per reverse fanout (7))
1. `srun` sends step create request to `slurmctld`
2. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O and launches tasks
8. On task termination, `slurmstepd` notifies `srun`
9. `slurmstepd` sends step completions to `slurmctld` (per reverse fanout (7))
10. `slurmctld` sends step completion to `slurmddbd`
Step congestion

- Step management is done by the controller
  - Can be a source of congestion
- Step management requires controller job write lock
  - Locks up the system
  - Slurm is highly threaded but not highly concurrent
- Bigger issue when creating many steps
  - Doing own resource management within allocation
  - E.g. 1 allocation with 1000's of steps
Step relief

- Move step management out of the controller
- Step management done by stepd (stepmgr)
- Reduce rpc congestion and locking on the controller
Stepmgr

- Designate a stepd as the stepmgr for the job
  - batch step / extern step / interactive step
- Job allocation/environment tells which slurmd has the stepmgr
  - STEPMGR_INTERFACE=<host>
- Stepmgr creates and manages steps for the job
Stepmgr

```
#!/bin/bash
srun hostname
```
Stepmgr

- `sbatch -N2`
- `ctld`
- `create allocation`
- `create batch step / extern step / interactive step`
- `pass job_record_t to slurmd, stepd`
- `slurmd n1`
- `batch step / stepmgr`
- `1. create step`
- `STEPMGR_INTERFACE=batch_host`
- `#!/bin/bash`
- `srun hostname`
Stepmgr

sbatch -N2

cltd

create allocation

create batch step / extern step / interactive step
pass job_record_t to slurmd, stepd

slurmd n1

2. slurmd connects to stepd/stepmgr

batch step / stepmgr

1. create step

3. stepmgr creates step credential

STEPMGR_INTERFACE=batch_host

#!/bin/bash
srun hostname
Step Launch Sequence

1. `srun` sends step create request to `stepmgr`
2. `stepmgr` responds with step credential
Step Launch Sequence

1. `srun` sends step create request to `stepmgr`
2. `stepmgr` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
Step Launch Sequence

1. `srun` sends step create request to `stepmgr`
2. `stepmgr` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
Step Launch Sequence

1. `srun` sends step create request to `stepmgr`
2. `stepmgr` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O and launches tasks
**Step Launch Sequence**

1. `srun` sends step create request to `stepmgr`.
2. `stepmgr` responds with step credential.
3. `srun` opens sockets for I/O.
4. `srun` forwards credential with task info to `slurmd`.
5. `slurmd` forwards request as needed (per fanout).
6. `slurmd` forks/execs `slurmstepd`.
7. `slurmstepd` connects I/O and launches tasks.
8. On task termination, `slurmstepd` notifies `srun`.
9. `slurmstepd` sends step completions to `stepmgr` (per reverse fanout (7)).
Step Launch Sequence

1. `srun` sends step create request to `stepmgr`
2. `stepmgr` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forwards request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O and launches tasks
8. On task termination, `slurmstepd` notifies `srun`
9. `slurmstepd` sends step completions to `stepmgr` (per reverse fanout (7))
10. a. no step accounting
    b. `stepmgr` sends step completion to `slurmdbd`
    c. `stepmgr` sends to slurmctld, slurmctld sends to `slurmdbd`
Things to tackle / figure out
Accounting

- Currently, the stepds reverse fanout (7) to stepd 0
  - stepds send to the controller if can’t reverse
- Controller sends info to dbd
- But controller doesn’t have record of steps
- Options
  - No step accounting
  - Accounting through slurmstepd
  - Accounting through slurmcctl
Heterogenous jobs

- The controller keeps track of the different jobs in a het job
  - thus the het step id.
- Controller to do step management for het jobs
Clients

- `squeue -s`
- `scontrol show steps`
  - query all slurmds and stepds to get list of steps?
Questions or Thoughts?