High Throughput Computing with SLURM

SLURM User Group Meeting
October 9-10, 2012
Barcelona, Spain

Morris Jette and Danny Auble
[jette,da]@schedmd.com

Thanks to

This work is supported by the Oak Ridge National Laboratory Extreme Scale Systems Center

Historic Perspective

- High Throughput Computing (HTC) was not a priority of original SLURM developers, so little effort had been made to optimize SLURM performance
  - Typical cluster ran only a few thousand jobs per day
- SLURM throughput (about 120 jobs/second) was already competitive with other schedulers, so the potential for additional throughput was unclear

* - Actual results may vary depending upon hardware and configuration

Initial Profiling Results

- Started by using *gprof* profiling tool to see where time was being spent in *slurmctld* and *slurmd* daemons
- Some time was going to the expected places
  - Sorting jobs by priority
- Some surprises
  - Functions that are relatively fast, but executed very frequently took much of the time
  - Plugin initialization check: executed for every plugin call
  - Time formatting

Plugin Initialization Check Logic

- Before

```
        lock()
        if (already_initialized)
          goto fini;
        /* initialize_plugin */
fini:   unlock();
        if (var)
           free(var);
        return;
```

These functions are called so frequently that a subtle change like this had a substantial effect

- After

```
        if (already_initialized)
             return;
        lock()
        if (already_initialized)
           goto fini;
        /* initialize_plugin */
fini:   unlock();
        if (var)
           free(var);
        return;
```

Unnecessary Time Formatting

- Before

```
char buf;
make_time_str(time, buf, size);
debug3("time=%s", buf);
```

Avoid time formatting by default in frequently used functions, especially if not typically used (e.g. debug3).

- After

```
#ifdef DEBUG_TIME
        char buf;
        make_time_str(time, buf, size);
        debug3("time=%s", buf);
#else
        debug3("time=%u", time);
#endif
```

Results of Phase 1

- Relatively minor changes to about 20 places in the SLURM code resulted in 300 percent speedup (from 120 jobs/second to 500 jobs/second)
- Many of these changes are included in SLURM v2.4 and benefit most SLURM configurations
- Remaining bottlenecks are very difficult to parallelize

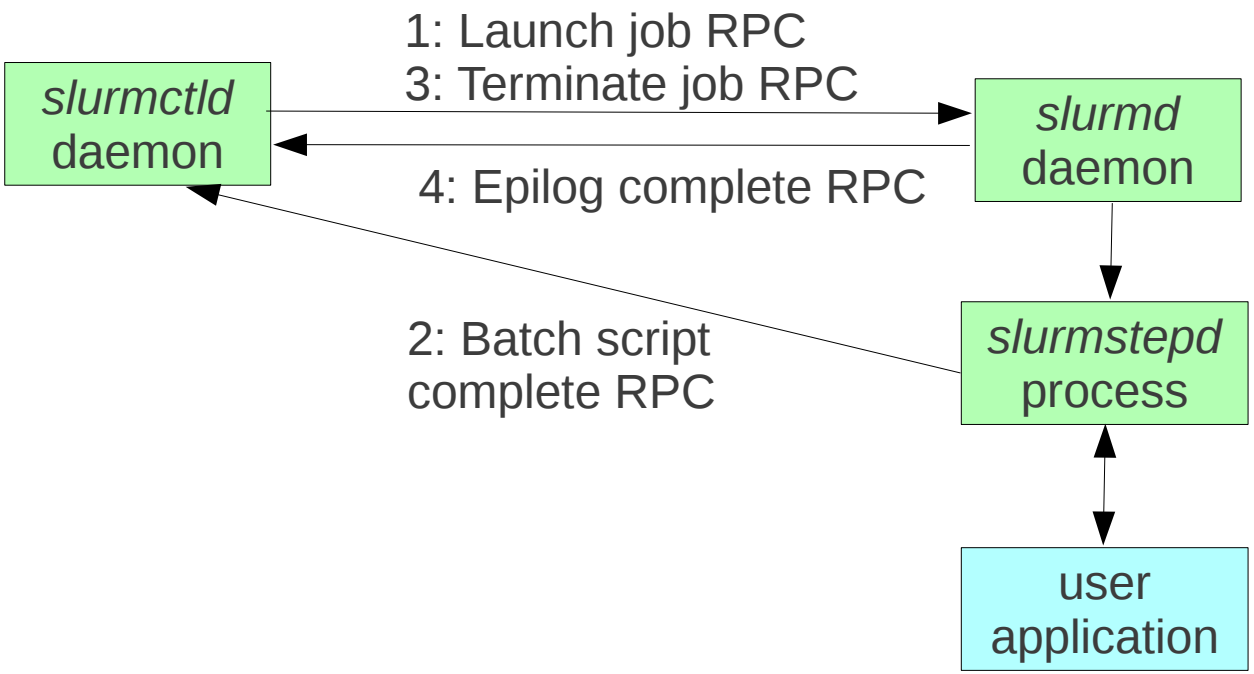
Phase 2: Configuration Specific Enhancements

- Serial (single CPU) jobs
  - Streamlined *select/serial* plugin
  - Compute node “pull” model
- FIFO scheduling

New *select/serial* plugin

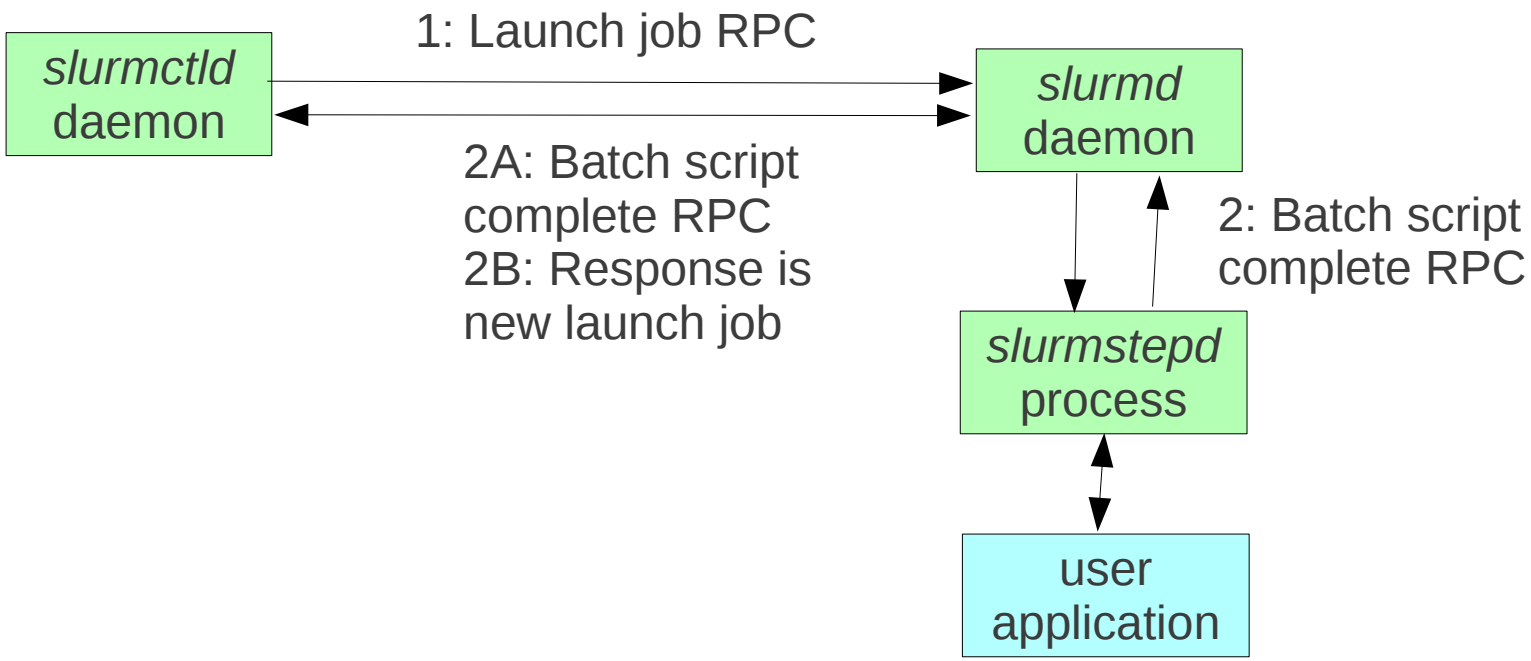
- SLURM's “select” plugin selects resources for a job
- Much code is used to select the “best” available resources (e.g. allocate multiple cores on a single socket rather than spreading a job across multiple sockets, optimized network topology, etc.)
- *Select/serial* is streamlined version of *select/cons_res*
  - Roughly half of the logic has been removed
- If all jobs use only a single processor, the “select” plugin execution time is measurably reduced, throughput up to 570 jobs per second

Batch Job RPC Sequence



Steps 1 & 2 happen on node zero of job allocation
Steps 3 & 4 happen on every node of a job allocation

Batch Job “Pull” Model



This model applies only to serial jobs
Batch script complete RPC sent from *slurmstepd* to *slurmd* (instead of *slurmctld*)
Slurmd executes epilog before notifying *slurmctld* that the batch script is complete
Response to batch script complete RPC is a new job launch request
4 RPCs reduced to 1 RPC in typical scenario
Throughput up to 600 jobs per second

FIFO Scheduling

- Eliminate all job sorting logic
- Use simple FIFO list in all scheduling logic
- With all changes, up to 630 jobs per second

How Fast SLURM Run?

- Remove all data structure locks
  - Jobs, nodes, partitions, etc.
  - This will result in data corruption; use for testing only
- Reaches 1000 jobs per second throughput

Results

Original SLURM v2.3	120 jobs/second
General logic improvements in SLURM 2.4	500 jobs/second
New *select/serial* plugin	570 jobs/second
New job “pull” logic	600 jobs/second
Job queuing logic enhancements (FIFO)	630 jobs/second
Without locks (results in memory corruption)	1000 jobs/second

Configuration Options

- These options will not be acceptable in all environments
- Purge completed jobs from *slurmctld* daemon as soon as possible
  - *MinJobAge=2*
- *Minimize logging*
  - *SlurmctldDebug=1*
  - *SlurmdDebug=1*
- Disable accounting
  - *AccountingStorageType, JobAcctGatherType, JobCompType*
- FIFO scheduling (eliminates priority ordering)
  - *SchedulerType=sched/builtin*

Possible Future Enhancements

- Substantial additional speedup with one *slurmctld* daemon not likely
- Multiple *slurmctld* daemons, say one per compute node, could offer much higher throughput
  - Each job submitted to specific compute node
  - Schedule each node independently
  - Requires all jobs fit entirely within one node
  - Job dependencies probably not supported
  - Job accounting probably not supported

Meta-cluster Model