JSM Overview
—
David Solt, Ph.D.
HPC Software Architect
Presentation Goals

Give a broad overview of JSM

Present basic concepts and functionalities

Be available for assisting with complex use cases
Topics

- Basic concepts of JSM
- JS utilities
- Resource Sets
- Simple layouts
- Advanced layouts
- Binding and OpenMP
- MIMD support
JSM is...

- Launcher
  - like srun, prun, mpirun, blaunch, etc

- PMIx server
  - MPI apps do not need to create an MPI daemon

- Sub-resource manager
  - LSF is the high level RM, but jsrun manages a users resources
Basic concepts

mpirun launch flow:
- Determine hosts given to you by resource manager
- Tell mpirun to use some subset of those hosts to run your job

jsrun launch flow:
- Describe the resources you want to JSM
- JSM runs your job on the resources it chooses to meet your criteria.
JSM utilities

**jsrun** - create a job step (or reservation)

**jslist** - list running, completed or killed job steps

**jskill** - signal a job step

**jswait** - wait for the completion of a job step
Resource Sets

jsrun defines bundles of resources (CPU, GPU & memory)

- called resource sets

- each resource set will result in a cgroup (unless cgroups are turned off)

- jsrun allocates CPU’s as physical cores (i.e. 44 cores per box - core isolation)

Why resource sets?

- Allows multiple jsruns to divide up resource on a node

- Simple way to describe the resources available to each rank

- Simple way to enforce locality between ranks
The basics

How many resource sets to create:
- `-n, --nrs <#|ALL_HOSTS>` How many resource sets to create

How many CPUs:
- `-c, --cpu_per_rs <#|ALL_CPUS>` How many CPU’s in each RS

How many GPUs:
- `-g, --gpu_per_rs <#|ALL_GPUS>` How many GPU’s in each RS

(Memory can also be assigned, but is not enforced)
Simple examples

- `jsrun -n ALL_HOSTS -c ALL_CPUS -g ALL_GPUS` ...
  - create a job step with all the resources in the entire allocation
  - resources grouped by node

- `jsrun -n 64 -c 6 -g 1` ...
  - create a job step with 64 resource sets each with 6 cpus and 1 gpu
Two job steps at once

Assume 2 nodes, 40 CPU's each (core isolation), 6 GPU each

Want to run 2 job steps at the same time that each use 1/2 the resources:

- jsrun --nrs 2 -c 20 -g 3 a.out
- jsrun --nrs 2 -c 20 -g 3 b.out

What do you expect will happen?
Option A:
Option B:
Option C: Chaos!
What will you get?

Option A is JSM's preferred allocation

Options B and C are possible

- but in reality would only be given if other jobs were running that had segmented the CPU/GPU space.

Option A or B can be requested
Influencers of resource sets - RS per host

- `--rs_per_host` Specifies the number of resource sets on each host

  - `jsrun -r 1 --nrs 2 -c 20 -g 3 a.out`
  - `jsrun -r 1 --nrs 2 -c 20 -g 3 b.out`
  - Will force scenario B (-r 2 will force scenario A)

- `--latency_priority=<comma separated list>`

  - Priorities are `cpu-cpu`, `gpu-gpu`, `mem-mem`, `mem-gpu`, `mem-cpu`, `gpu-cpu`, `CPU-CPU`, `GPU-GPU`, `MEM-MEM`, `MEM-GPU`, `MEM-CPU`, `GPU-CPU`.

SMPI / OLCF Readiness Workshop / March 5, 2018 / © 2018 IBM Corporation
Influencers of resource sets - latency_priority

- `l`, `--latency_priority=<comma separated list>`
  - priorities are `cpu-cpu`, `gpu-gpu`, `mem-mem`, `mem-gpu`, `mem-cpu`, `gpu-cpu`, `CPU-CPU`, `GPU-GPU`, `MEM-MEM`, `MEM-GPU`, `MEM-CPU`, `GPU-CPU`.
  - Default set by configuration file
  - Capital letters: Only resources which are optimal for the given priority will be accepted (wait for other steps to finish if necessary)
  - Lower case: Use the resources which are best available at the time
  - Default default is: `gpu-cpu, cpu-mem, cpu-cpu`
--latency_priority options

- Select CPUs from the same socket
- Select CPUs & GPUs from the same socket
- Select CPUs and memory from same NUMA
- Select GPUs that are from the same socket
- Select GPUs & memory that are from the same socket
- Select memory that comes from the same NUMA

When do you actually care?
- jsrun --nrs <x> -c 12 -g 6
- Do you prefer CPU’s close or CPU’s to GPU’s close?
Save, keep or use or re-use an allocation

Allocations can be created and saved (-A, --allocate_only)
- `jsrun --allocate_only --nrs 8 -c 6 -g 6 --rs_per_host 1`

Allocations can be used immediately and then removed
- `jsrun --nrs 8 -c 6 -g 6 --np 8 a.out`

Allocations can be used/removed but saved for later recreation:
- `jsrun --save_resources myoptimal16noderun.txt --nrs 8 -c 6 -g 6 --np 8 a.out`
- `jsrun --use_resources myoptimal16noderun.txt --np 8 a.out`
- `jsrun --use_resource myoptimal16noderun.txt -A`

Saved allocations can be re-used (-J, --use_reservation)
- `jsrun --J 1 --np 8 a.out`
Creating Tasks

How many tasks to create

Where to place them

How many cores to assign to each task
How many tasks

-p, --np <x>

- Create x tasks

-a, --tasks_per_rs <x>

- Create x tasks per resource set

(Nothing)

- Get 1 task per resource set by default
Simple layouts

1 task per core on all nodes:
- `jsrun a.out` (with default config file)
- `jsrun -c 1 --nrs X --np X a.out`

1 task per 4 cores, 1 GPU each:
- `jsrun -c 4 -g 1 --nrs X --np X a.out`

8 tasks per core:
- `jsrun -c 1 --nrs X --np $((X*8)) a.out`

1 task per core, 3 GPU’s per task, 1 task per node:
- `jsrun -c 1 -g 3 --rs_per_node 1 --nrs X a.out`
Where to place tasks

-d, --launch_distribution <packed|cyclic|plane:<x>>

- How to map tasks to resource sets.
- Do you want contiguous ranks in each resource set?
- Do you want non-contiguous ranks in each resource set?

- NOTE: In GA release, packed will pack ranks based on how many cores each rank is assigned (See --bind packed:<x>)

-H, --launch_node_tasks=<#>

- Schedule the specified task number on the launch node
How many cores to reserve for each task

Binding: how many and which cores to bind to a task
-b, --bind=<none, rs, packed[:<#>]>  
   - none - don’t bind  
   - rs - bind to the entire resource set  
   - packed:<n> - bind each task to at most n cores. Choose cores that are closest to each other.
<table>
<thead>
<tr>
<th>--bind option</th>
<th>cores bound to</th>
<th>OMP_PLACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>1 core per task</td>
<td>ALL SMT threads from one core</td>
</tr>
<tr>
<td>none</td>
<td>none (though cgroup will still limit cores if cgroups are on)</td>
<td>1 task per RS: All SMT threads in RS &gt; 1 task per RS: unset</td>
</tr>
<tr>
<td>rs</td>
<td>all cores in the RS</td>
<td>All SMT threads in RS</td>
</tr>
<tr>
<td>packed:&lt;n&gt;</td>
<td>n cores from the RS</td>
<td>ALL SMT threads from n cores</td>
</tr>
</tbody>
</table>
Advanced layouts

1 task per core, each 4 tasks share 1 GPU:
- jsrun -c 4 -g 1 --a 4 a.out

1 task per core, 8 tasks communicate via shared memory and need to be close to one another:
- jsrun -c 8 --a 8 --latency_priority CPU-CPU a.out

1 task per 4 cores, every 4 tasks share a GPU:
- jsrun -c 16 -g 1 --a 4 --bind packed:4 --launch_distribution packed a.out

Odd ranks on one host, even on another
- jsrun --nrs 2 -c ALL_CPUS --np X --launch_distribution cyclic a.out
MIMD support

jsrun -f <appfile>

Appfile:

<# ranks> : <reservation #> : <command>
<# ranks> : <reservation #> : <command>
<# ranks> : <reservation #> : <command>

etc.
sh-4.2$ jsrun --nrs 2 -c 4 -g 1 /bin/true

sh-4.2$ jslist -R

<table>
<thead>
<tr>
<th>parent</th>
<th>nrs</th>
<th>cpus</th>
<th>gpus</th>
<th>exit</th>
<th>ID</th>
<th>ID</th>
<th>nrs</th>
<th>per RS</th>
<th>per RS</th>
<th>status</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>Complete</td>
</tr>
</tbody>
</table>

RS 0 HOST c712f8n10:
  SOCKET 0: cpus: 0-3 gpus: 0 mem: 4000

RS 1 HOST c712f8n10:
  SOCKET 0: cpus: 4-7 gpus: 1 mem: 4000
Can be used to implement flow control between job steps.
Define resource sets (CPU’s & GPU’s)
(Use jslist -R to see what you got)
Determine number of tasks
Determine task distribution
Determine cores per task
Environmental influencers

-h, --chdir=<path>
- Change current working directory.
-i, --immediate
- Force jsrun to return immediately.
-L, --use_spindle=<0|1>
- Should spindle be used
-M, --smpiargs=<SMPI args>
- Quoted argument list meaningful for Spectrum MPI applications
-P, --pre_post_exec=<script info>
-X, --exit_on_error=<0|1>
- Determine if a rank error should result in namespace abort
-D, --env_no_propagate=<var>
- Exclude this environment variable from being propagated
-E, --env=<var=val>
- Environment variable to be set before exec of tasks
-F, --env_eval=<var=val>
- environment variable to be evaluated and set before exec of tasks
stdio related options

- `e`, `--stdio_mode=individual | collected | prepended`
  - Individual: Every rank writes to its own local file
  - Collected: IO goes through jsrun
  - prepended: collected + rank identification on each line

- `stdin_rank=<#>`
  - Collected mode only. Only one rank may receive stdin in collected mode.

- `k`, `--stdio_stderr=<filename>`
  - stderr filename (default: jsrun for collected, /dev/null for individual)

- `o`, `--stdio_stdout=<filename>`
  - stdout filename (default: jsrun for collected, /dev/null for individual)

- `t`, `--stdio_input=<filename>`
  - stdin filename (default: jsrun for collected, /dev/null for individual)