

# Ensemble Job Submission on the Blue Gene/Q: The Right Tool for the Job

Paul Rich

May 20, 2014





# Overview

- Definitions
- Picking the right type of job
- Basic script mode job tips
- Ensemble script job tips
- Subblock script job tips

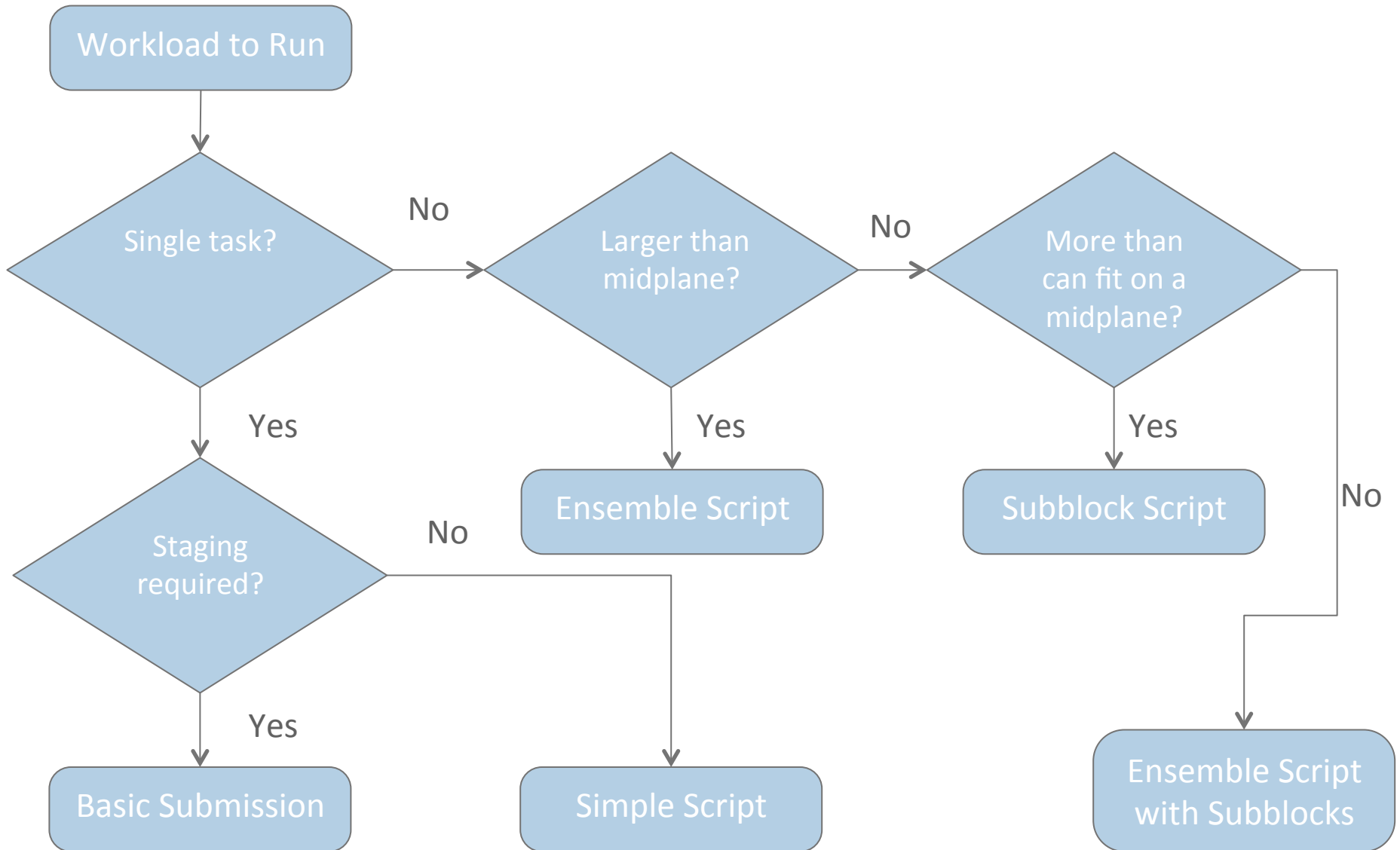
# Definitions and disambiguation

- Cobalt Job -- A job submitted to Cobalt via qsub. Shows up in qstat
- Blue Gene Job -- A task run on the Blue Gene compute nodes via runjob
- Script Job -- A Cobalt job submitted with the *--mode script* option
- Ensemble Jobs -- A Cobalt job that allocates a block from the scheduler and boots multiple blocks simultaneously from this allocation
- Subblock Jobs -- A runjob feature where multiple Blue Gene jobs are run on the same booted block. May be combined with ensemble jobs within a script job.

# Types of script jobs and the best tool for the job

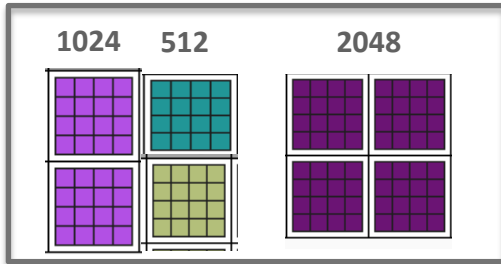
- Basic Script Jobs
  - You have a task to run and some minor staging that you wish to have occur automatically
  - You need to prompt the system to take extra actions after your run
  - You have a small series of short tasks that can run on the same hardware, and want to minimize boot time
- Ensemble Jobs
  - You want to run multiple simultaneous tasks on smaller blocks within a larger allocation
  - You want to change block size between tasks
- Subblock Jobs
  - Runjob feature provided by IBM
  - You have a number of small tasks to run
  - All tasks are smaller than the smallest block size on the system
- Neither of these are MPMD
- Ensemble Jobs and Subblock Jobs are not either-or

# Choosing the right type of submission



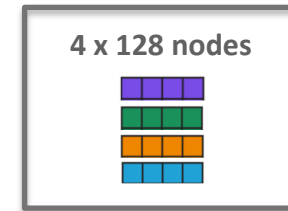
# Examples of ensemble and subblock jobs

Example of ensemble jobs

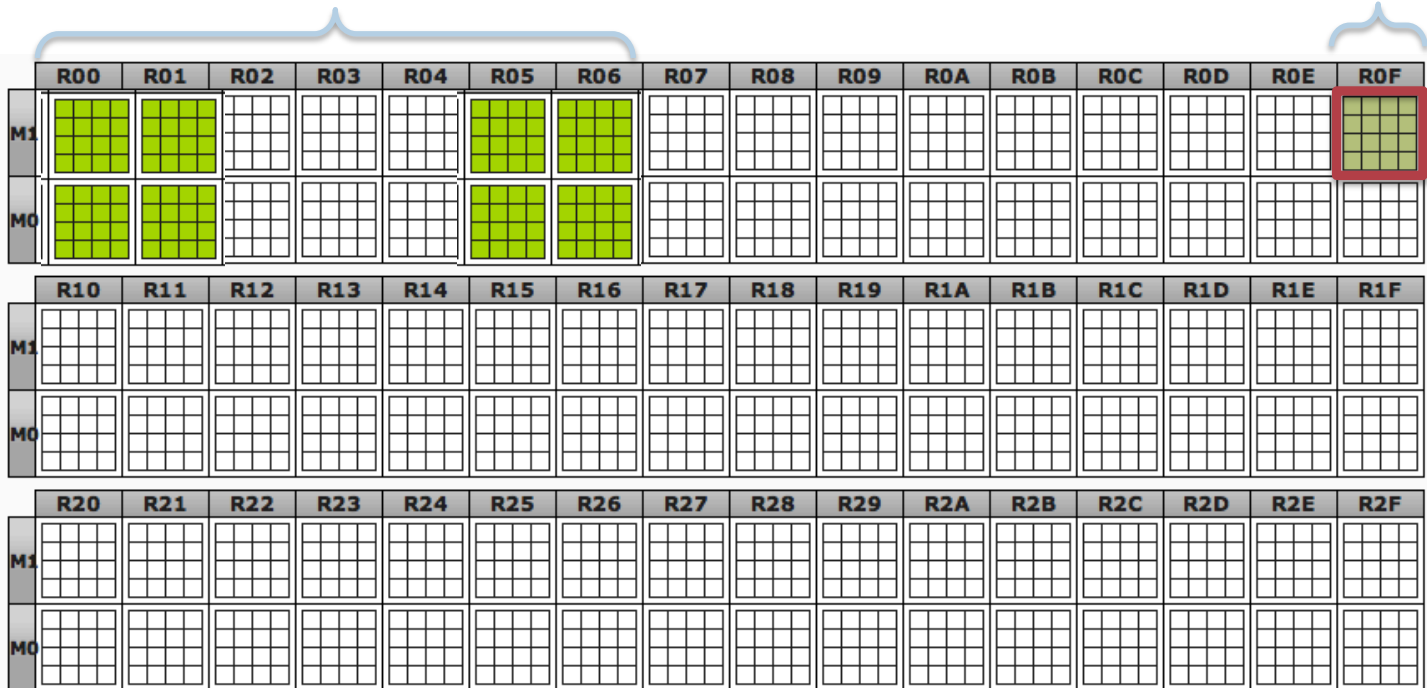


4K

Example of subblock jobs



512 nodes



Minimum partition size on Mira

For jobs with the same characteristics: higher job size = faster score increase



# General script job advice

- The job is charged for the set of allocated compute resources for the entire runtime.
  - Do not run expensive operations like compiles on the script host if you can help it.
- Check Exit and Block Statuses between runs
- Do not delete Cobalt-generated files as a part of the script.
  - This includes the .cobaltlog, and .error files.
  - Help us help you.
- Do be careful about what you run last during a script.
  - “echo done” will cause a script to always have an exit status of 0 regardless of what else has happened!
  - Consider using the ‘-e’ flag if using a shell script.
  - Very important if you’re using job dependencies.

# Setting up a basic script job

- Submit with *--mode script* on your qsub line
- Script can be anything executable on a *front end node*
- Allocated block will be booted before the start of the script
- Use Cobalt-provided variables when possible: \$COBALT\_JOBID, \$COBALT\_PARTNAME, \$COBALT\_PARTSIZE, etc.
- Invoke runjob from your script. You may run multiple tasks on the same block multiple times in series
- You may have to use the *boot-block --reboot* command between runs if:
  - Partlist shows your block as having a “SoftwareFailure”
  - Your program exited with a non-zero exit status
- If using BG\_PERSISTMEMSIZE, remember that contents will not persist past reboots.



# Advanced script jobs: ensemble jobs

- Running multiple tasks concurrently
- Block either must start off unbooted or be freed at the start of the job
  - Disable block booting by setting `--disable_preboot` in your `qsub` line
- May run on any “child” block of your allocated block
  - May be subject to wiring restrictions
- *get-bootable-blocks* utility will get all blocks that are available to boot in your allocation
  - Can constrain to particular sizes and geometries
  - Calls after a boot will not include blocked blocks
- *boot-block* boots, frees or reboots a particular location. When this utility completes, the block should be ready for use.
  - If a nonzero exit status is returned, a problem occurred with the boot.
- `Runjob` works exactly the same way, just using one of the child locations per invocation.

## Example ensemble script

```
#!/bin/bash
BLOCKS=`get-bootable-blocks --size 512 $COBALT_PARTNAME`
for BLOCK in $BLOCKS
do
    boot-block --block $BLOCK &
done
wait
for BLOCK in $BLOCKS
do
    runjob --block $BLOCK : ./my_binary &
done
wait
for BLOCK in $BLOCKS
do
    boot-block --block $BLOCK --free &
done
wait
```

*From <http://trac.mcs.anl.gov/projects/cobalt/wiki/BGQUserComputeBlockControl>*

# Ensemble script caveats

- Some geometries may have issues running together.
  - Notably, a problem for 4096 node blocks and 1024 node full-torus blocks due to physical wiring.
  - Make sure to hit get-bootable-blocks again if you're using these!
- Boots may fail:
  - File systems may fail to mount. Hardware may die during boot.
  - Recommend a maximum of three retries
- Software errors can be cleared by rebooting
  - If running multiple jobs, partlist will show an error as blocked (SoftwareFailure)
- Can mix sizes and reboot as different sizes, but reboot required to switch.
- Once a block is booted, can run multiple runjobs against it.
- Some blocks may share IO resources so check ALCF system documentation.
- Test your script on Cetus, if possible.

# Running with subblocks

- Subblock jobs may be used within any script job
- Must target booted blocks of 512 nodes or smaller
  - Can run down to the single-node level
- Requires the use of the `--corner` and `--shape` flags to `runjob`
- Corner must be a hardware location
  - Can obtain this from a coordinate from `/soft/cobalt/bgq_hardware_mapper/coord2hardware`
  - Use the first 5-tuple of the block name for the origin
  - Groups of corners may be obtained by passing the block name and shape to `/soft/cobalt/bgq_hardware_mapper/get-corners.py`
- Shape are the lengths of each dimension
  - `man runjob` has a list of common shapes for valid subblock sizes
- A compute block going into error does not kill previously running jobs
  - Will prevent future jobs from starting
- Watch out for overloading IO nodes
- May be run in the same script as an ensemble job



# Questions?

# Partition dimensions on BG/Q systems

## Cetus

Nodes	A	B	C	D	E
128	2	2	4	4	2
256	4	2	4	4	2
512	4	4	4	4	2
1024	4	4	4	8	2

## Vesta

Nodes	A	B	C	D	E
32	2	2	2	2	2
64	2	2	4	2	2
128	2	2	4	4	2
256	4	2	4	4	2
512	4	4	4	4	2
1024	4	4	4/8	8/4	2
2048	4	4	8	8	2

## Mira

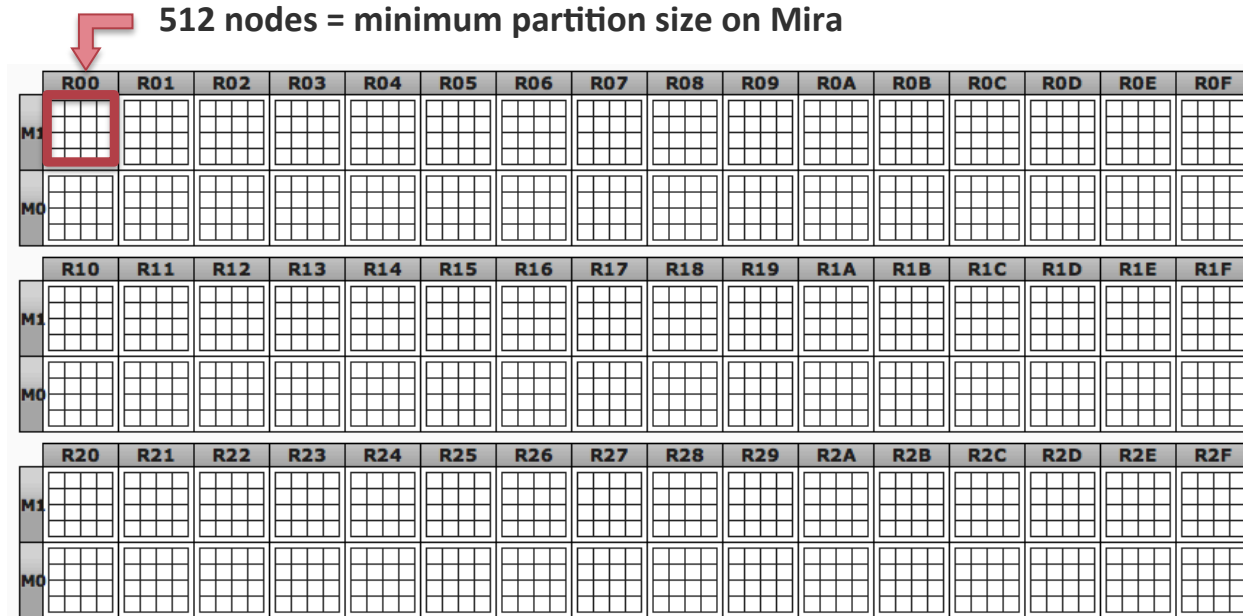
Nodes	A	B	C	D	E
512	4	4	4	4	2
1024	4	4	4	8	2
2048	4	4	4	16	2
4096	4/8	4	8/4	16	2
8192	4	4	16	16	2
12288	8	4	12	16	2
16384	4/8	8/4	16	16	2
24576	4	12	16	16	2
32768	8	8	16	16	2
49152	8	12	16	16	2

**Command: partlist**

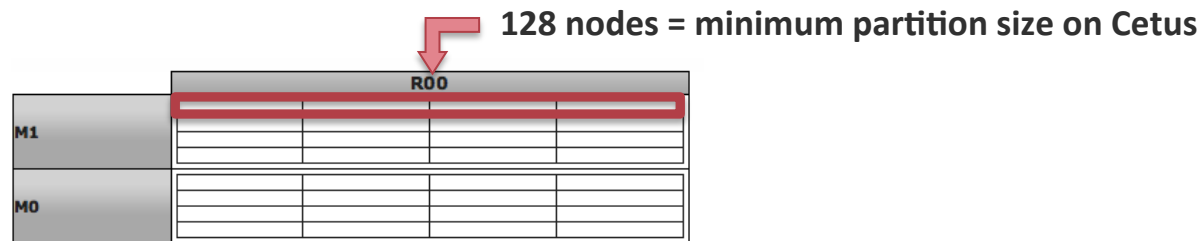
<http://www.alcf.anl.gov/user-guides/machine-partitions>

# Minimum partition sizes on BG/Q machines

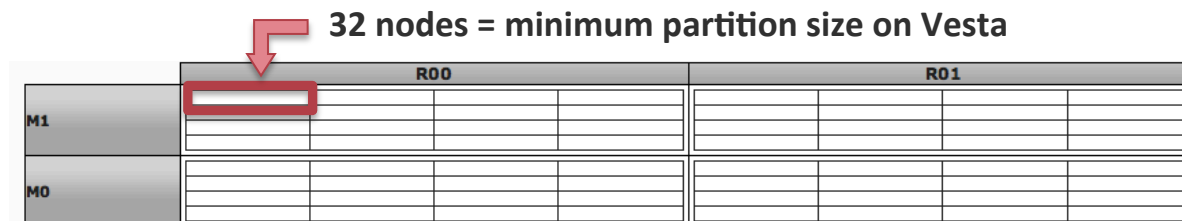
**Mira**  
48 racks



**Cetus**  
1 rack



**Vesta**  
2 racks



# Block translation made easy

- `/soft/cobalt/bgg hardware_mapper` contains basic helper scripts
- `hardware2coord` -- take a hardware location and translate to ABCDE
- `coord2hardware` -- take an ABCDE location and translate to a hardware location
- `get-corners.py experimental` -- given a block name and a shape, generate every valid `--corner` argument for that shape on that block
  - Must be used on a block of 512 nodes or smaller