pawsey

Supercomputing User Training

Module 10: Testing Job Runs

Pawsey Training Series

Supercomputing User Training

1. Supercomputing Introduction
2. Logging In
3. Filesystems Overview
4. Moving Data In and Out
5. Using Software Modules
6. Using Software Containers
7. Accounting Model Overview
8. Job Scheduling Overview
9. Running Jobs
10. Testing Job Runs
11. Managing Project Data

Outcomes for this Module

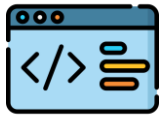
- Describe why we need to test job runs
- Describe how to optimise resource requests
- Discuss job runtime versus total cost

✓ Prerequisite knowledge:

- ✓ **User Training 07: Accounting Model Overview**
- ✓ **User Training 08: Job Scheduling Overview**
- ✓ **User Training 09: Running Jobs**

Watch for These Signs!

Definition of new concepts



Hands-on coding (demo)

Best practices

Exercises and solutions

Warnings (bad practices)

Links to user documentation

Icons made by Freepik from www.flaticon.com

Pawsey

Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

Testing Job Runs

Why is it Useful to Test Job Runs?

- Before running a production job for the first time
    1. To make sure both the scheduler and the program configurations are correct and as needed
    2. To optimise resource requests, in particular wall time, node/task/cpu count, memory

- In preparing a merit application
    3. To produce data that demonstrate scaling of your scientific workflow

NOTE: use the debug partition for test runs whenever applicable, to reduce waiting times in the queue

1. Testing a Job for the First Time

- Goal: making sure the job can run as intended

- Verify the contents of the script
  - Scheduler directives
  - Shell commands
- Verify any other input file
- After the production job starts, monitor the output in real time, to make sure it is going as intended
- In alternative, to reduce waiting time for the test, configure a test job to run for fraction of total simulation length, and optionally smaller problem size

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

2. Optimising Resource Requests

- Maximum duration: identify the optimal one
  - Underestimating will result in the job not completing
  - Overestimating may result in longer waiting in queue
  - Run job with maximum wall time, watch how much it takes, and adjust accordingly
  - In alternative, configure job to run for fraction of total simulation length and/or linearly smaller problem size, and proportionally extrapolate duration for full run
  - Reassess after optimising for job size (next two slides)
- Memory requirements
  - Start with default
  - If job runs out of memory, increase memory request progressively, until it runs successfully

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

2. Optimising Resource Requests

- Whenever possible, test on fraction of total simulation length
- Size of job: MPI applications
  - Focus on number of tasks and nodes
  - Start from 8 tasks in 1 node (one L3 cache domain)
  - Progressively double up the number of tasks in 1 node
    - ntasks = 16, 32, 64, 128
    - ntasks-per-node = 16, 32, 64, 128
  - Double up beyond one node
    - ntask s= 256, 512, ..
    - ntasks-per-node = 128
  - Record
    - Runtime of computationally intensive job task
    - Total cost = runtime * number of used cores (tasks)
  - See example in second next slide

```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

2. Optimising Resource Requests

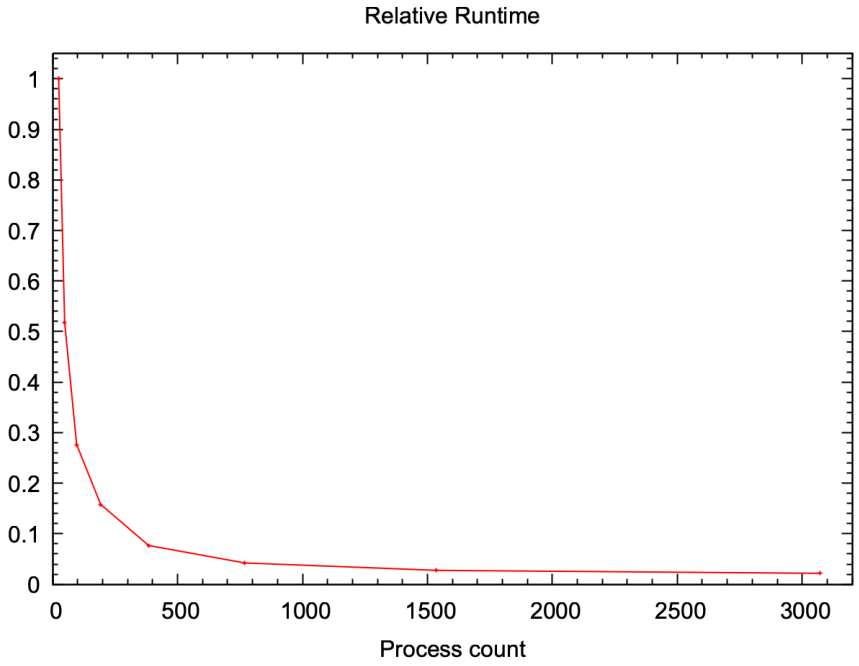
- Whenever possible, test on fraction of total simulation length
- Size of job: OpenMP applications
  - Focus on number of cpus per task
  - Start from 1 cpu per task
  - Progressively double up the number of cpus in 1 node
    - cpus-per-task = 2, 4, 8, 16, 32, 64, 128
  - Record
    - Runtime of computationally intensive job task
    - Total cost = runtime * number of used cores (cpus)
    - Both can be computed relative to a reference run
  - See example in next slide

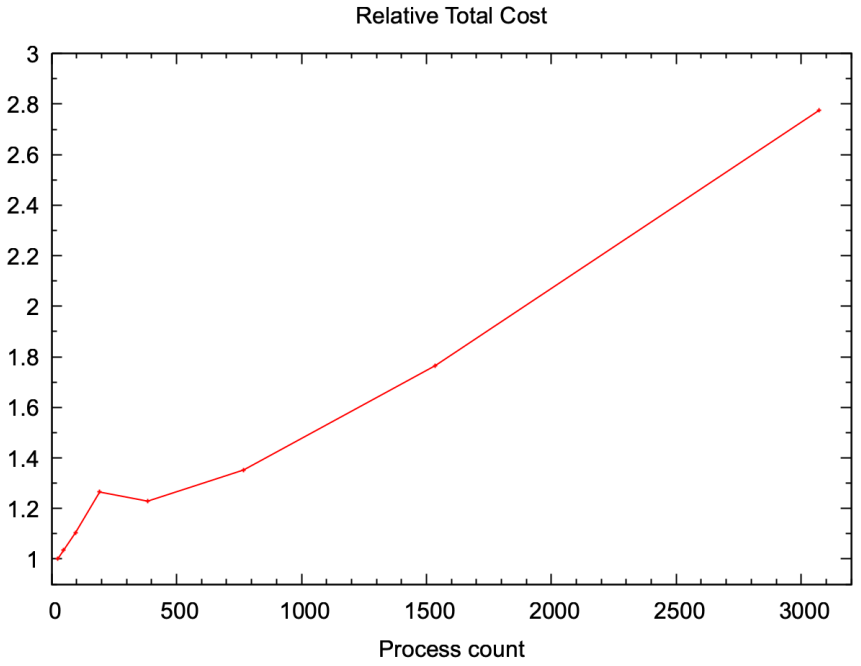
```
#!/bin/bash -l

#SBATCH --job-name=useful-name
#SBATCH --account=project-id
#SBATCH --partition=partition
#SBATCH --ntasks=processes
#SBATCH --ntasks-per-node=procs-per-node
#SBATCH --cpus-per-task=threads-per-proc
#SBATCH --mem=memory-per-node
#SBATCH --time=max-duration

..

srun ./program
```

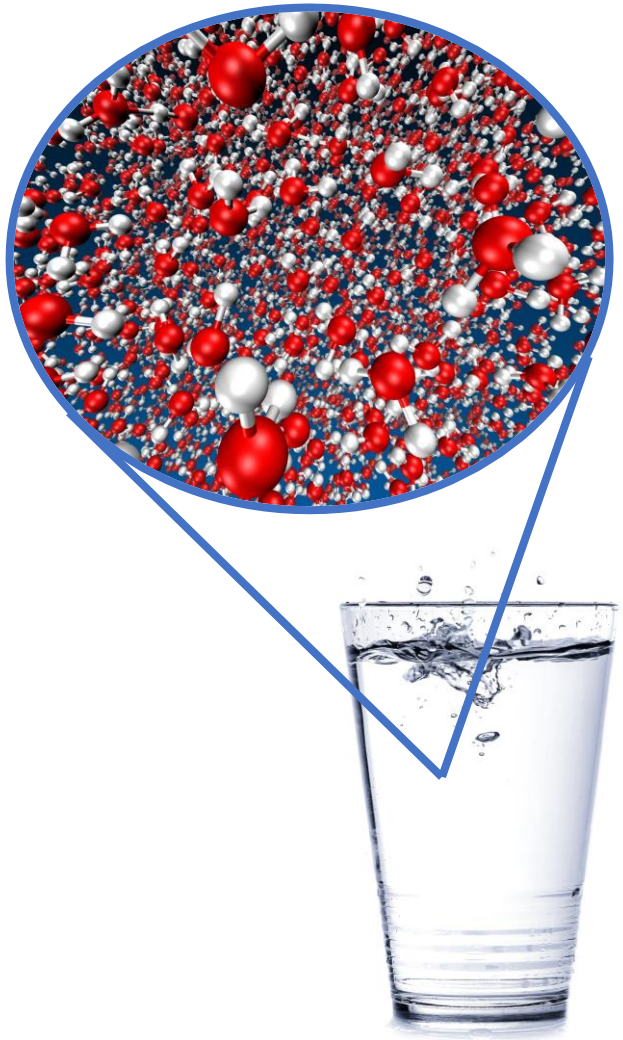
2. Optimising Resource Requests: Runtime and Total Cost

MPI example: Molecular Dynamics of a Water box with 800,000 atoms with LAMMPS

Reference run: 24 processes (1 node) ; Data available within the [materials on Github]







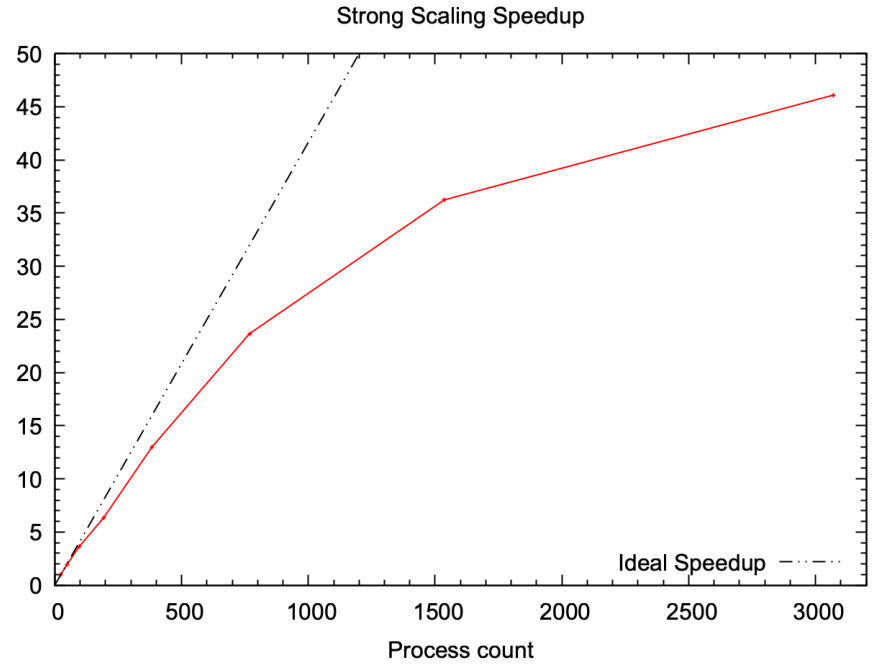
Here is the point:

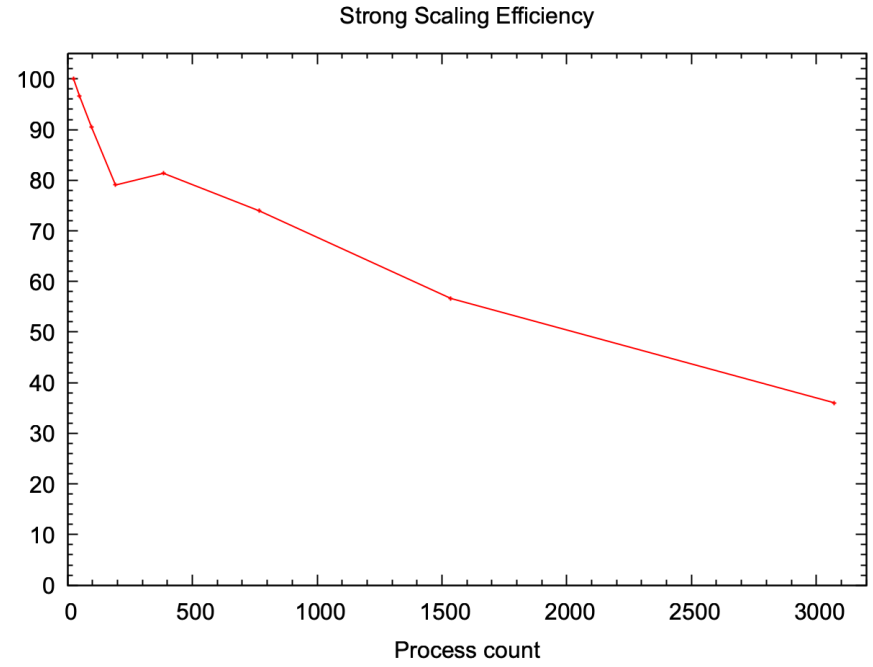
- Parallelism comes at a cost
- How much extra-cost (in accounted service units) are you willing to take, to get your outputs faster?

3. Demonstrating Scaling

- Goal: near-ideal scaling at high core count
- Two types of scaling
  - Weak: problem size increasing with core count
  - Strong: same problem size, increasing core count
    - Strong scaling is less trivial to achieve
- Record two measures of scaling
  - Speedup = Inverse of relative runtime
    - Relative to a reference run
  - Efficiency = Speedup / Ideal Speedup

MPI example: Molecular Dynamics of a Water box with 800,000 atoms with LAMMPS ; Reference run: 24 processes (1 node)





Summary

- Testing job runs can be useful
  - Before running a production job for the first time
  - In preparing a merit application
- Resource requests typically to be optimised
  - Maximum duration
  - Memory requirement
  - Job size (number of cpus/tasks/nodes)
- Consider job runtime vs total job cost, depending on urgency of your runs
- Provide evidence of good scaling at high core count for your software, to make your merit application stronger

Pawsey

Australian Government

NCRIS National Research Infrastructure for Australia

CSIRO

Curtin University

Murdoch University

Government of Western Australia

ECU Edith Cowan University

The University of Western Australia

Getting Help

Getting Help

https://support.pawsey.org.au

Pawsey has extensive User Support Documentation.

Areas covered include:

- System user guides
- Knowledge Base
- Pawsey-supported software list
- Maintenance logs
- Policies and terms of use

For further assistance, contact the help desk, via User Support Portal.

Help us to help you by providing details, such as:

- Which resource
- Error messages
- Location of files
- SLURM job id
- Your username if having login issues
- Never tell us (or anyone) your password!

Become a Pawsey Friend and receive our Newsletter:
https://pawsey.org.au/pawsey-friends/

Q & A Session