Data Management

File-systems on ARCHER

Introduction

- Archer like many HPC systems has a complex structure
- Multiple types of file system
  - Home
  - Work
  - Archive
- Multiple node types
  - Login
  - Compute
  - Serial batch/post-processing
  - Data transfer

Home

- Four file-systems unified view as /home
  - /home1
  - /home2
  - /home3
  - /home4
- Approx 60TB each
- Standard Network-Attached-Storage (NAS)
- Backup supported
- Compilation and interactive tools
  - Not intended for high-performance or large data-sets.

Directories in /home

- Every project has an allocation on one home file-system
- Your home directory will live here
  - /home/*<project>*/*<group>*/*<username>*

Work

- Three file systems, unified view as /work
  - /fs2 1.5 PB
  - /fs3 1.5 PB
  - /fs4 1.8 PB
- High performance parallel file-system build using **lustre**
- Main working disks for parallel jobs
  - Only file-systems available on the compute nodes so binaries and data files should live here

Directories in /work

- Every project has a directory-tree in one of the /work file-systems
    - **/work**/*<project>/<group>/<username>*
- Projects can create different groups to manage space allocation if they want.
- If you need to share data within a group use
    - **/work**/*<project>/<group>/***shared**
- If you want data to be readable outside the project use
    - **/work**/*<project>/***shared**

Quotas

- Allocations are set using group quotas.
- Group quota limits total amount of space taken by files with a particular group-id.
  - Each file can only be in one group.
- Default group for files.
  - Follows group of directory if “**s**” flag set on directory.
  - Otherwise effective group of user.
- The default group permissions are set so that group-id should default correct branch of the directory tree.
  - Unless changes using **chgrp, rsync tar** etc.

Archive (The RDF)

- Three distinct file-systems
  - /epsrc 1.1 PB epsrc projects
  - /nerc 3.9 PB nerc projects
  - /general 235TB others
- Parallel file-systems built using **gpfs**
- Independent of ARCHER (part of the Research Data Facility)
  - General storage infrastructure. File-systems can grown.
- Tape Backup to second site.
- Primarily intended as a safe-haven for important data.
  - Though performance is not bad either
- Similar directory structure to /work
  - Allocation by file-set

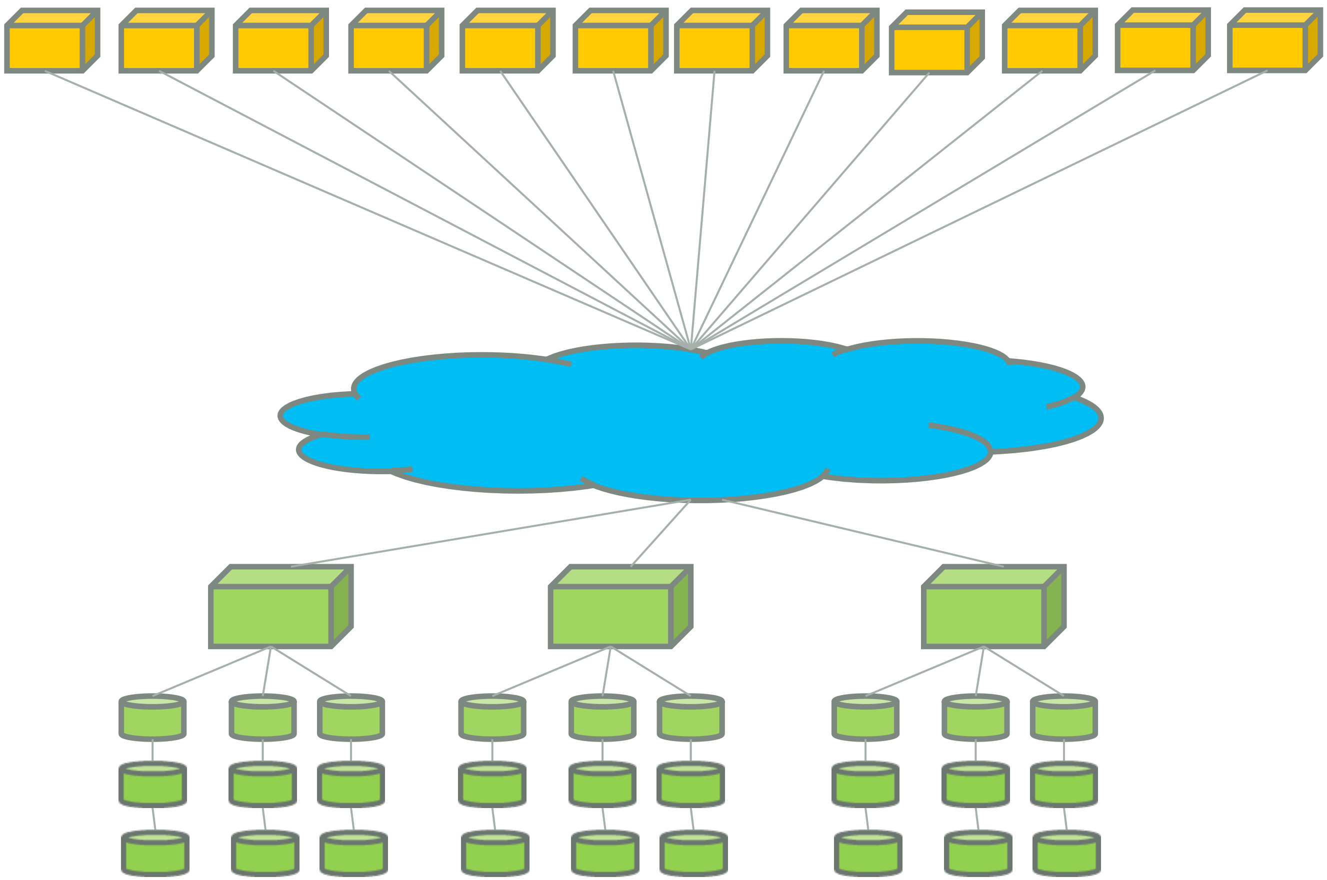
Mount locations

	home	work	archive
login	☑	☑	☑
compute	☒	☑	☒
Serial batch/pp	☑	☑	☑
Data transfer	☒	☒	☑

Parallel File systems

- All these file-systems are built out of similar disk technology.
- Lustre and GPFS support larger and more performant file-systems by using parallelism
  - Large numbers of disks in parallel.
  - Hosted by multiple file-servers.
  - Serving multiple clients
- Max performance is roughly proportional to capacity.

Overview



High performance IO

- Key to good performance is to use the hardware in parallel
- Large single files will be striped across multiple disks/ servers
  - Need to use large block transfers otherwise you will only be hitting one disk at a time.
  - Single stream performance will still be limited by cpu, memory and network bottlenecks for the single client compute node.
  - Much better to have multiple compute nodes perform IO at the same time.

Meta-data

- Meta-data operations are much less parallel than data transfers
  - File open/close
  - Directory listing
  - Query file-size/ownership/modify-date etc.
- This can be a serious problem when large numbers of compute nodes keep performing meta-data operation.

Metadata good practice

- Don’t check for existence before opening
  - Open will return error if file does not exist.
- Keep files open rather than close/reopen
- Consider routing IO through a sub-set of nodes.
- Very large numbers of small files will be slow to process later so use parallel IO rather than separate files for each MPI rank.
- File length check very expensive on lustre
  - Needs to query each parallel server
- Have independent clients operate in different directories
  - Less lock contention.
  - Better meta-data caching.

File system contention

- Remember file-systems are a **shared** resource.
- Performance will depend on what other users are doing at the same time.
- This is particularly true of meta-data operations.
- Reducing contention increases performance and reduces timing variability.

Moving data around

- Standard unix tools can be used to move data around
  - **cp**
  - **rsync**
  - **dd**
  - **cpio**
- Can be run interactively on login nodes for short simple cases but often better to submit a serial batch job.
- Don't perform data movement as part of a parallel batch job as you are still being charged for the parallel nodes while the copy takes place.

Chaining batch jobs

- If may want to use serial batch jobs to perform pre/post processing for a parallel job. Several options:

1. Submit follow-on jobs from the batch script
   - Simple but follow-on job starts at the bottom of the queue.
   - Probably ok if second job is serial
2. Use job dependencies:

```
$ qsub first.pbs
12345.sdb
$ qsub –W depend=afterok:12345.sdb second.pbs
```

Increasing parallelism

- May be some advantage to breaking up large data movements into multiple batch jobs.
- Copy different directories in parallel using different serial jobs.
- Or run multiple background copies from one script

```
cp –r directory1 dest &
cp –r directory2 dest &
wait
```

- Jobs will contend with each other but might still complete quicker.
  - Best if clients are working on different directories

Impact of metadata operations

- File metadata operations and lack of parallelism can make large numbers of small files particularly expensive to manipulate.
- If you generate data like this consider packing related data into larger archive files.
- Compare copying 200000 1KB sized files with a single large file on lustre. Note timings very variable due to contention.

	dd bs=500k	cp -r	rsync -a
200000 x 1K files	NA	10m 14s	1h 55m 9s
1 x 200000K file	0.46 s	0.34 s	1.08s

rsync

- Popular and powerful data movement tool.
  - Can synchronize directories
  - Can be set to only move missing/changed files
  - Supports data transfers between hosts
- Unfortunately this adds some overhead and requires ***more*** meta-data operations so rsync can be slower than a simple copy.

Lustre tuning

- Lustre provides **lfs** command to control/query file-system behaviour:

```
spb@eslogin008:/work/z01/z01/spb> lfs getstripe bigdata
bigdata
lmm_stripe_count:   4
lmm_stripe_size:    1048576
lmm_layout_gen:     0
lmm_stripe_offset:  33
      obdidx        objid        objid        group
          33     39337975    0x2583ff7            0
          25     39349614    0x2586d6e            0
          17     39605440    0x25c54c0            0
           1     39611128    0x25c6af8            0
```

Striping

- Data is striped across Object Storage Targets (OST)s
- Default is to stripe each file across 4 OSTs
  - Blocksize approx 1MB
  - Reasonable default for single stream access.
- Default striping can be changed using **lfs setstripe** before creating file
- Worth increasing for large files read/written in parallel.
- A **lfs cp** command supports optimised copy
  - Only for copies within lustre though.
  - Does not seem to help with metadata problems.

Reusing this material