



CANCER  
RESEARCH  
UK

CAMBRIDGE  
INSTITUTE

# CRUK CI HPC cluster introduction (III of III)

## Some advanced topics



UNIVERSITY OF  
CAMBRIDGE

# Reference Genomes

- Path to reference data:  
`/scratchb/bioinformatics/reference_data/reference_genomes/`
- Path to assembly:  
*.../organism/assembly/*
- What we maintain:
  - Genome sequence (fasta)
  - Alignment indices: BWA, TopHat, Bowtie (1,2)
  - Annotations:
    - GTF format gene model
    - RefFlat format gene model
    - Signal artifact list (if available)

# Working with Lustre

1. Revisit architecture
2. Stripes
3. Avoiding I/O Bottlenecks
4. Using System Cache

## Lustre: Quick Review

Lustre is a massively parallel distributed file system

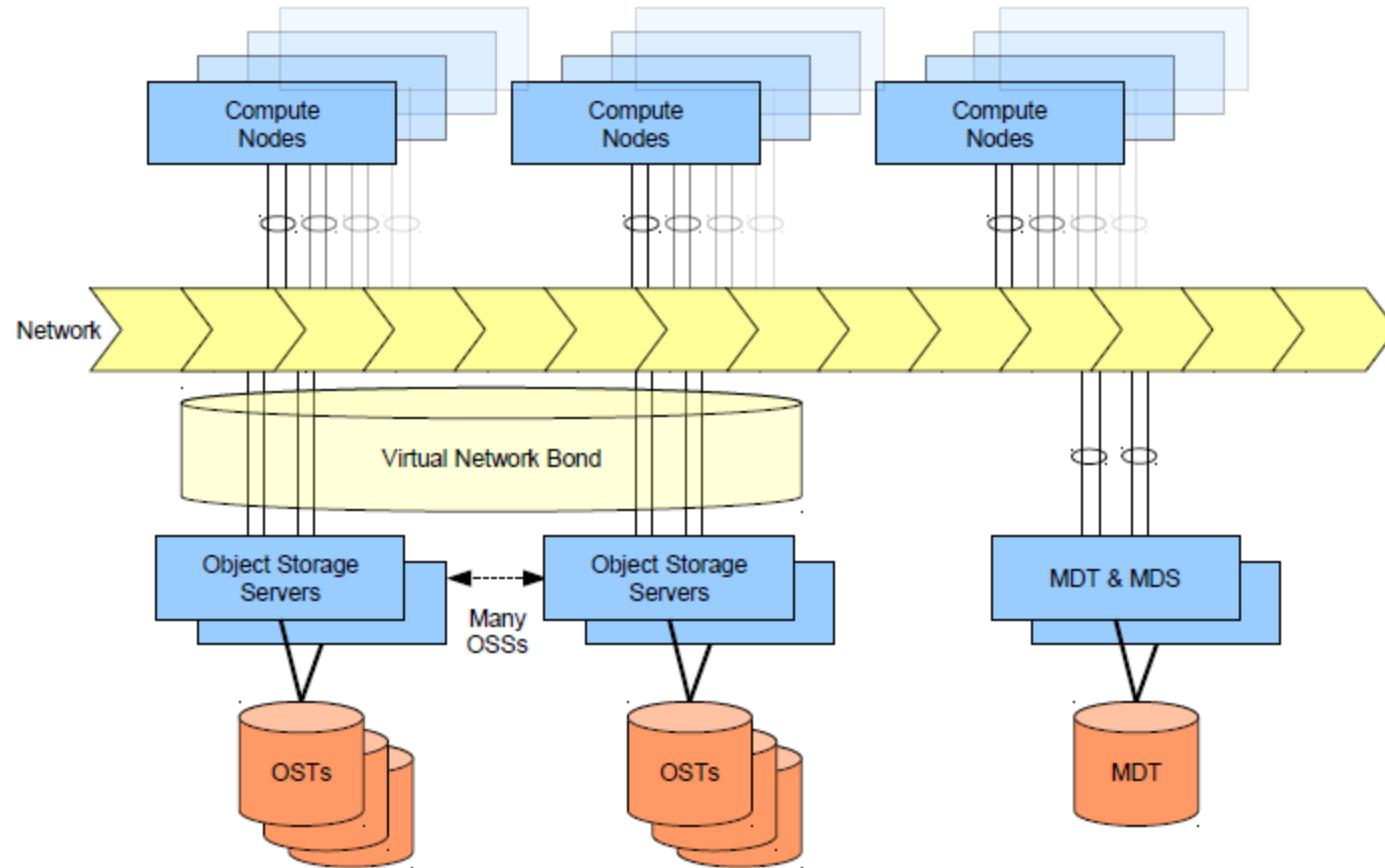
- Deployed in 7 out of 10 most powerful supercomputers
- POSIX compliant

Lustre design paradigm concepts

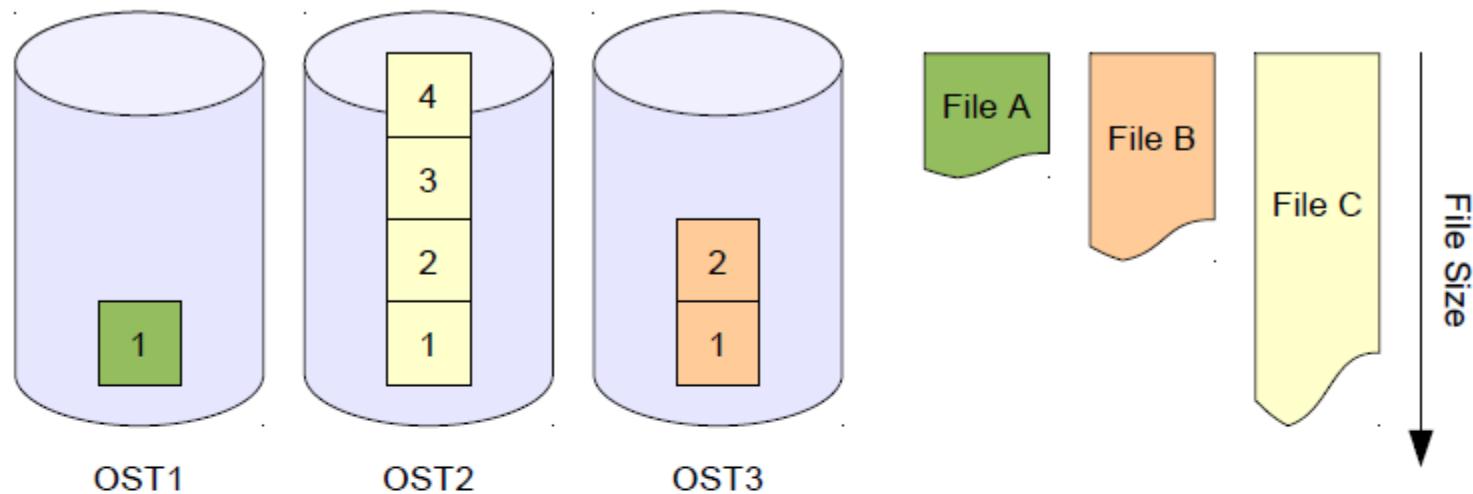
- Separation of file meta-data and storage allocation
- Scalable data serving through parallel data striping
- Aggregates network bandwidth
- Distributed operation

'scratch' storage we (deliberately) don't back it up

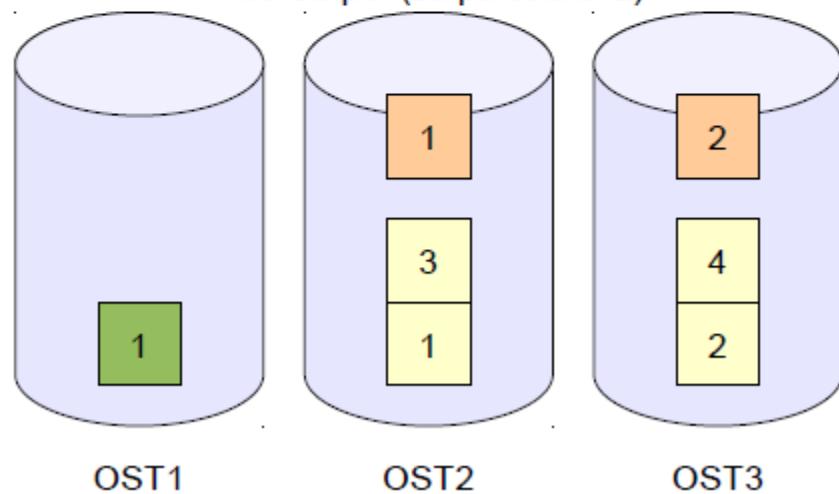
## Lustre Architecture ...



Single Stripe (stripe count=1)



Two Stripe (stripe count=2)



...and File Striping

# File Striping of Large Files

## Performance benefits

- Aggregates I/O bandwidth to single large file
- In general, more stripes improves performance
- Small overhead associated with open/closing striped files

(Striping allows file size to exceed single OST size)

## Many jobs reading single file

- For example blastdb and maq reference data
- `/lustre/reference_data/genomes`

## Many jobs reading & writing multiple large files

- Requires benchmarking

## Many jobs writing to single file

- High bandwidth but requires careful coding (can be disastrous)

## Set Stripe Information

Set per file or directory

Default is not to stripe

Only newly created files will be stripes

- Use cp (not mv) to migrate existing files

```
clust1-headnode ~ $ lfs setstripe <file|dir> --size <stripe_size>
--count <count> --index <index>
```

Where,

size = stripe size specified in k, m or g (0 default 1MB)

count = OST stripe count (0 defaults 4 OST and -l over all OSTs)

index = OST index of first stripe (-l indicating default)

## Read Stripe Information

Inspect file and directory stripe information  
with **lfs getstripe**

```
Clust1-headnode ~ $ lfs getstripe -d /scratchb/reference_data/genomes
stripe_count: 65535 stripe_size: 0 stripe_offset: 0

Clust1-headnode ~ $ lfs getstripe -d /scratchb/reference_data
stripe_count: 1 stripe_size: 1048576 stripe_offset: 0

Clust1-headnode ~ $ lfs getstripe /scratchb/reference_data/genomes/Zea_mays/zma.3.ebwt
lmm_stripe_count: 16
lmm_stripe_size: 1048576
lmm_stripe_offset: 11
obdidx objid objid group
11 623006 0x9819e 0
0 8504376 0x81c438 0
...
12 622252 0x97eac 0
8 607894 0x94696 0
```

# Using System Cache

Disk access is slow (no escape from this!)

- Memory access measured in a few nanoseconds
- Disc access measured in 10s of milliseconds

Linux uses free memory as cache

- Memory reclaimed as least used files expunged

“Pre-warming” cache

- Can increase I/O performance

```
uk-cri-lcst01 ~ $ cat largefile > /dev/null  
uk-cri-lcst01 ~ $ grep searchString largefile
```

# Avoiding Cache misses

Say you want to compare 3 sequences against 3 large databases:

“Out of the box” example

- sequence 1 vs database 1 **No cache: disk read required**
- sequence 1 vs database 2 **Cache miss: disk read required**
- sequence 1 vs database 3 **Cache miss: disk read required**
- sequence 2 vs database 1 **No cache: data expunged from cache**
- sequence 2 vs database 2 **Cache miss: disk read required**

## Re-ordering to avoid cache misses

- sequence 1 vs database 1 **No cache: disk read required**
- sequence 2 vs database 1 **Cache hit: data in cache**
- sequence 3 vs database 1 **Cache hit: data in cache**
- sequence 1 vs database 2 **No cache: disk read required**
- sequence 2 vs database 2 **Cache hit: data in cache**

# Avoiding Bad Performance

- Interactive use
  - Stattting files can be slow (common with shared file systems)
  - Avoid directly editing small files on lustre (keep to /home)
  - Turn off “color ls” (stat required for each file/directory)

## Random seeks

- Small random I/O extremely slow on lustre
- Avoid, as much as possible, running databases on lustre  
e.g. mysql, sqlite, Berkeley DB etc

## Limit number of files in directory

- 10,000s files in single directory bad (avoid, if possible)
- Use lfs setstripe to confine all files to single OST – obviously for small files only!

```
Clust1-headnode ~ $ lfs setstripe --count 1 directory
```

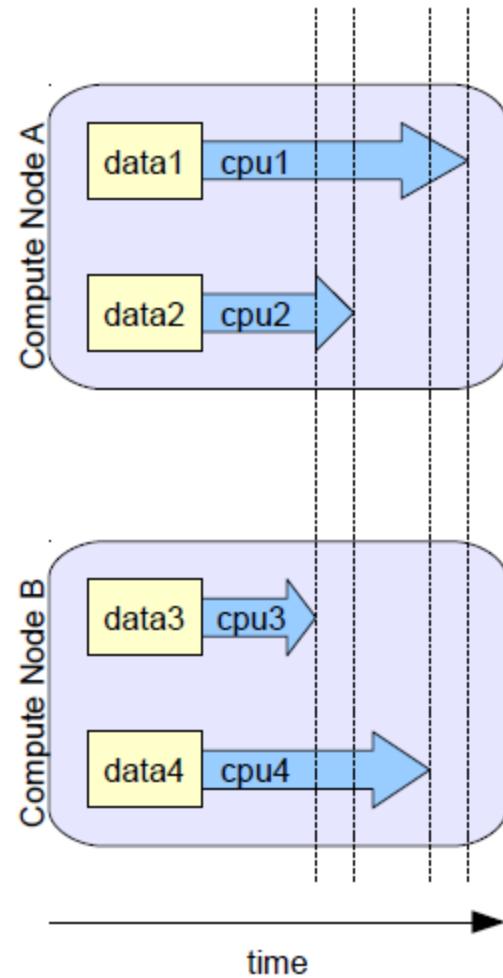
# Simple Parallel

(‘Embarassing’ or ‘Trivial’ in the computing science literature)

Solving many similar and independent tasks

- Analysis split into tasks
- Task assigned to one cpu
- No inter-task communication
- More throughput by running more tasks
- Task runtime varies

90% of bioinformatics codes fall into this model



# Shared Memory

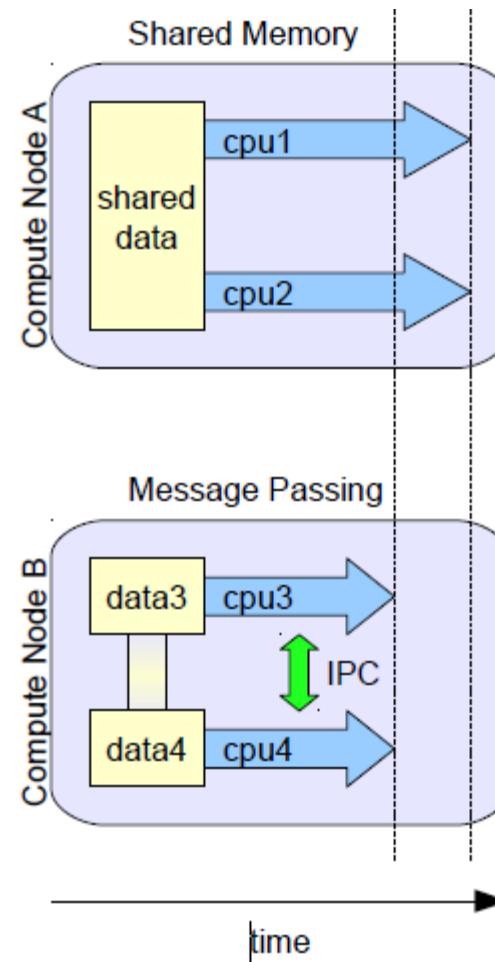
Shared tasks and memory

- Tasks assigned to cpus or cores
- Inter-task communication via shared memory
- Runtime decreases by adding more threads

# Message Passing - local

Multiple processes communicate using O/S level systems. Code must be specifically written to exploit parallelism

- OpenMP/OpenMPI/etc



# Message passing over network

Single task split across many compute nodes

- inter-machine communication (IMC)
- through MPI/OpenMP libraries again

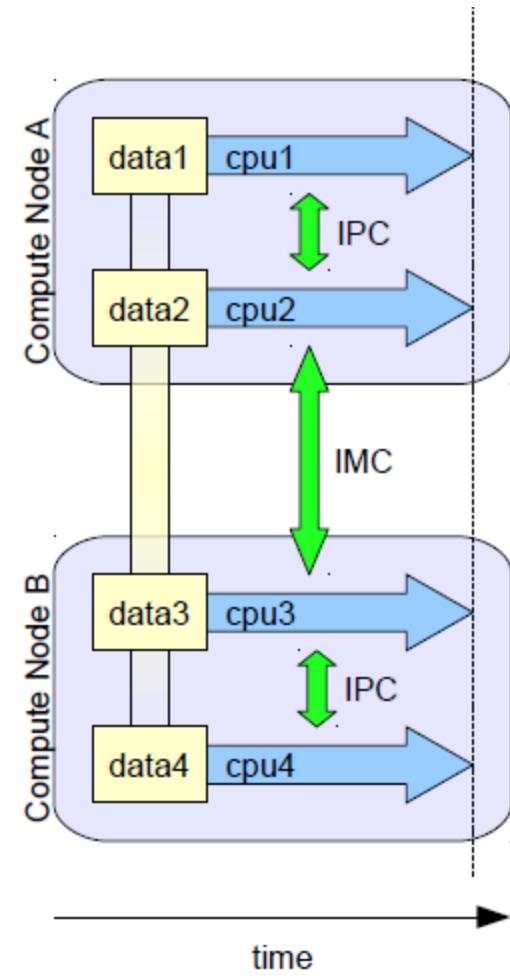
## Hybrid models

Single task split across many compute nodes

- Mix SMP, local MP, network MP

Can be tricky to predict performance.

Your code may get quite complex...



# Parallel Workloads

What happens when you increase your dataset size?

- $O(n)$ ,  $O(n \log n)$ ,  $O(n^2)$ ,  $O(n^9)$
- Runtime & memory increase ( $O$ ) with problem size ( $n$ )

Calculations are carried out in parallel

- Operating on principle that large calculations can often be divided into  $N$  smaller tasks
- These tasks are solved concurrently
- Time reduces to a function of  $O(n^x)/N$

## Parallel overheads

- communication, concurrency, parallel I/O introduce new overheads
- $O(N)$ ,  $O(N \log N)$ ,  $O(N^2)$

## Amdahl's Law

- Speed up of parallel application is ultimately limited by the fixed runtime of any sequential sections
- There's no magic bullet for this, you may have to change your algorithm, or mix parallel and single node sections.



CANCER  
RESEARCH  
UK

CAMBRIDGE  
INSTITUTE

## Acknowledgements

Marc O'Brien

Jon Marshall

Jing Su

Mark Fernandes



UNIVERSITY OF  
CAMBRIDGE