



INSTITUTO DE CIENCIAS MATEMÁTICAS

CSIC UAM UC3M UCM

Introducción a la Infraestructura de Computación del ICMAT

Programa

- Computación de altas prestaciones
- Recursos de computación de altas prestaciones
- La infraestructura de computación del ICMAT: Lovelace
 - Arquitectura física
 - Arquitectura lógica
 - Software
- Programación paralela
 - Lenguajes, herramientas, compiladores habituales, comparativa.

Acerca de mi

- Ingeniero en Informática
- 17 años de experiencia profesional en el sector privado, principalmente telecomunicaciones (Ono, Vodafone)
- Dedicado principalmente al área de operaciones e infraestructura, pero también con experiencia en desarrollo (C/C++, php, Python, .NET, ...)
- Amplia experiencia en servidores grandes y procesos de alta demanda de recursos. Programación paralela, clusters y grids, almacenamiento de altas prestaciones, etc.
- Desde diciembre de 2016, en el ICMAT dando soporte al área de Informática y de Computación del Instituto.

Email: alfredo.caso@icmat.es

Teléfono: 91 299 97 03

Computación de altas prestaciones

- ¿Qué es?

“Conjunto de tecnologías y técnicas que permiten la resolución de problemas computacionalmente complejos y/o costosos mediante la agregación de poder de cómputo”. (Mi definición)

- No es **únicamente** la utilización de servidores más potentes.
- Exige a menudo **aproximaciones diferentes** a la resolución de problemas (paralelismo, algoritmos complejos), así como hardware específico.
- No suelen existir estándares o soluciones generales válidas para todas las necesidades. A menudo, han de construirse **soluciones específicas** para cada necesidad.

Recursos HPC en el ámbito CSIC/UAM

- CCC: Centro de Computación Científica del Campus de Excelencia de la UAM <https://www.ccc.uam.es/>
- Red Española de Supercomputación (RES) <https://www.bsc.es/innovation-and-services/links-to-hpc-resources/access-to-res>

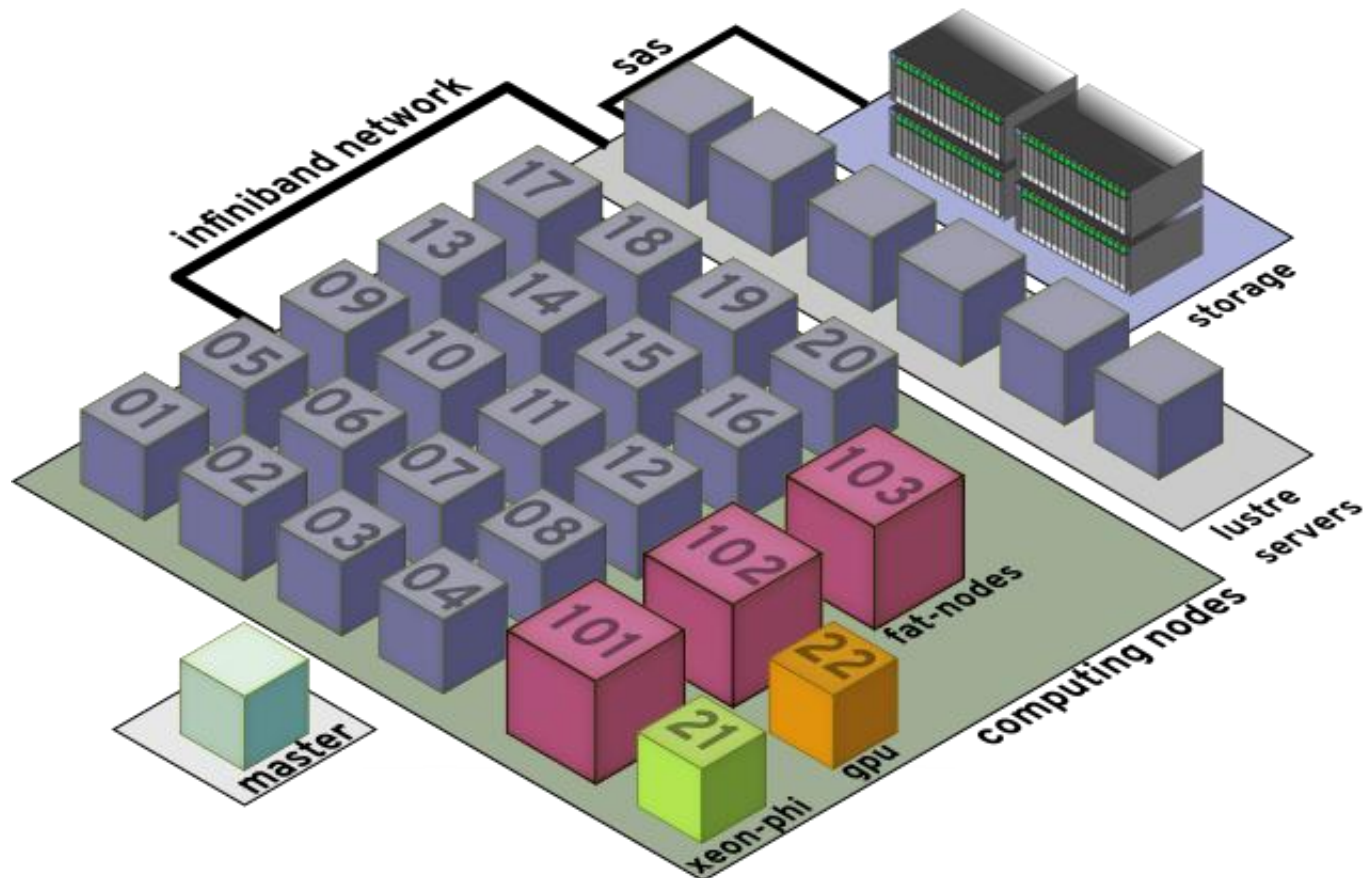


Comparativa de HPC (red RES)

Entidad	Computador	Tflops	CPU - núcleos	Memory (TB)	Storage (TB)	Network
BSC-CNS	Mare Nostrum	1.000,0	48.000,0	115,0	2.000,0	IB FDR10
CESVIMA	Magerit II	103,0	4.576,0	10,0	430,0	IB QDR
IAC	LaPalma			2,0	38,5	
IFCA	Altamira		1.500,0		900,0	Myrinet
UMA	Picasso	31,0	2.312,0	3,0	750,0	
Universitat de Valencia	Tirant	18,0	512,0	1,0	24,0	
CeSAr	Cesaraugusta	25,8	3.072,0			
UAM	Cibeles					
CSUC	Pirineus	14,3	1.344,0	6,1	112,0	
CenitS	Lusitania		2.800,0	2,0		
BSC-CNS	MinoTauro	180,0	1.500,0	3,0		IB QDR
UC	Altamira	80,0		0,3		IB FDR
ITC	Atlante	3,0		0,6	3,0	Myrinet
FCFS	Calendula	33,0	2.800,0	8,5		IB FDR
CESGA	FinisTerra	328,0	7.712,0	44,5	750,0	IB FDR
ICMAT	Lovelace	13,4	400,0	2,7	36,0	IB QDR

Fuente: el rendimiento de lovelace medido con Intel LINPACK + MKL, agregando el rendimiento de los nodos, las tarjetas XeonPhi y las tarjetas NVIDIA Tesla. El resto según la información en sus páginas web.

El cluster Lovelace. Arquitectura HW



Lovelace – Nodos de cálculo

20 nodos de
computación con:

2 CPUS [Intel\(R\) Xeon\(R\) CPU E5-2640 v3 @ 2.60GHz](#) (8 cores, 20 Mb L3
caché) [16 cores]
32 GB RAM de memoria a 1,8 GHz

3 nodos de computación
con:

2 CPUS [Intel\(R\) Xeon\(R\) CPU E5-2640 v3 @ 2.60GHz](#) (8 cores, 20 Mb L3
caché) [16 cores]
512 GB de memoria RAM a 1,8 GHz

1 nodo GPGPU con:

2 CPUS [Intel\(R\) Xeon\(R\) CPU E5-2640 v3 @ 2.60GHz](#) (8 cores, 20 Mb L3
caché) [16 cores]
256 GB de memoria RAM a 1,8 GHz
2 GPU [NVIDIA Corporation GK110GL \[Tesla K20m\]](#) (rev a1)

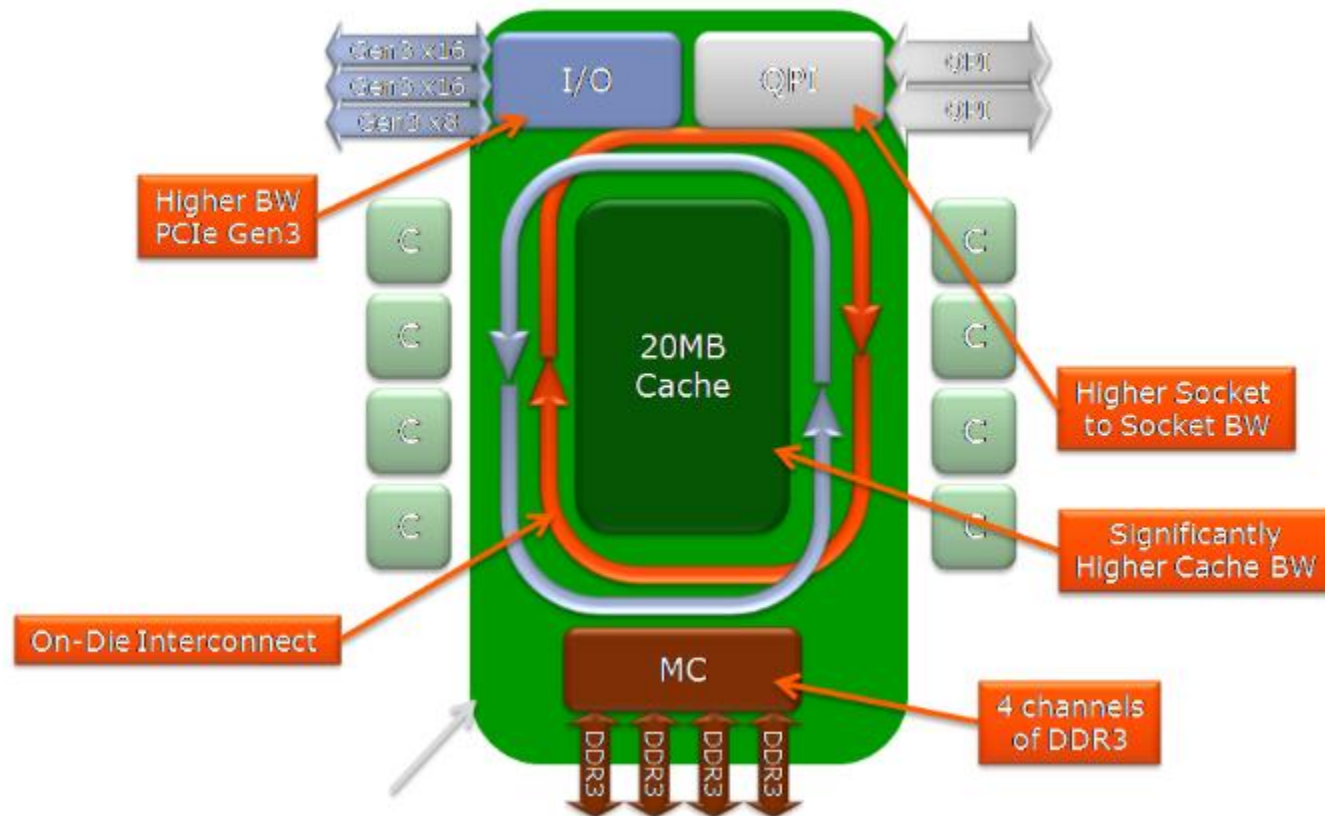
1 nodo Xeon-Phi con:

2 CPUS [Intel\(R\) Xeon\(R\) CPU E5-2640 v3 @ 2.60GHz](#) (8 cores, 20 Mb L3
caché) [16 cores]
256 GB de memoria RAM a 1,8 GHz
2 [Intel Corporation Xeon Phi coprocessor 5100 series \(rev 11\)](#)

CPU – latencias típicas - comparativa

Evento	ns	us	ms	Comparativa de escala
Referencia a caché L1	0,5	-	-	1 segundo
Fallo en predicción de rama	5	-	-	10 segundos
Referencia a caché L2	7	-	-	15 segundos
Mutex lock/unlock	25	-	-	Casi un minuto
Acceso a memoria principal (DDR3)	100	-	-	5 minutos
Comprimir 1KiB mediante zip	3.000	3	-	Más de hora y media
Enviar 1KiB sobre una red Gigabit	10.000	10	-	5 horas y media
Leer 4KiB aleatoriamente en un SSD	150.000	150	-	Casi dos días (~1GB/sec SSD)
Leer 1MiB secuencialmente de memoria principal (DDR3)	250.000	250	-	Casi tres días
Enviar un paquete de red ida y vuelta en el mismo datacenter	500.000	500	-	Casi seis días
Leer 1 MiB secuencialmente de SSD	1.000.000	1	1	(~1GB/sec SSD, 4X memory)
Búsqueda aleatoria en disco duro	10.000.000	10	10	Cerca de 4 meses
Lectura secuencial de 1MiB en disco duro.	20.000.000	20	20	Cerca de 8 meses
Enviar paquete de red ida y vuelta	150.000.000	150	150	Cerca de 5 años

Lovelace CPUs Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz



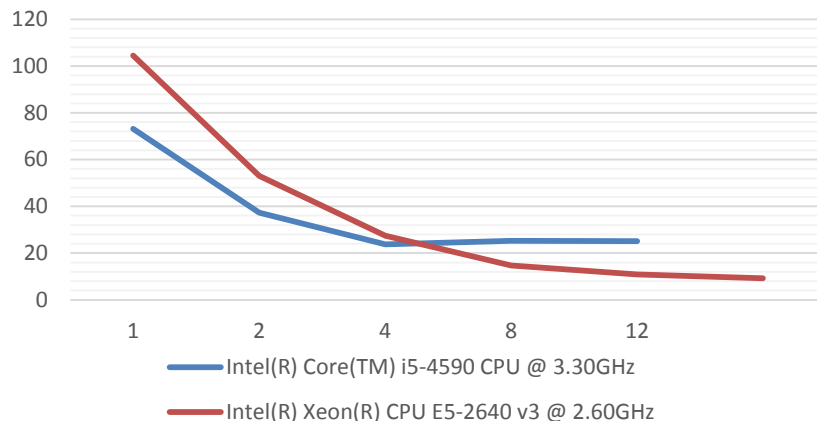
- Mejoras en cálculo de número flotante: Intel® Advanced Vector Extensions (Intel® AVX)
- Intel® Turbo Boost Technology 2.0
- High Bandwidth Last Level Cache
- High Bandwidth/Low Latency modular on-die Ring Interconnect
- Integrated Memory Controller with 4 channel DDR3

Frecuencia de CPU vs paralelismo

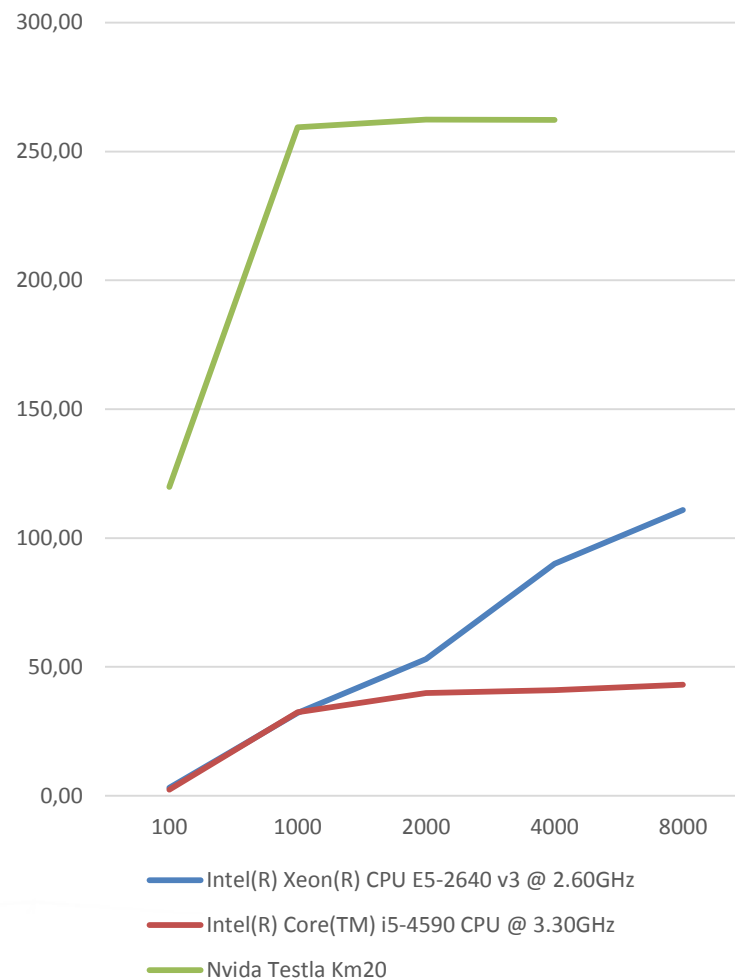
- En términos generales, para **1 único hilo de ejecución** (en serie), CPUs con menor número de núcleos pero mayor velocidad de reloj son mejores (mayor número de Gflops/s)
- Existe un límite práctico alrededor de los 5 Ghz (por ahora)
- Las CPU de los sistemas multi-core pueden:
 - Mejorar los tiempos de resolución de problemas **que puedan paralelizarse de forma eficiente**.
 - Soportar mayores cargas de usuario
 - Realizar varios trabajos **“a la vez”**
 - Mejorar los consumos eléctricos

Comparativa – multiplicación de matrices

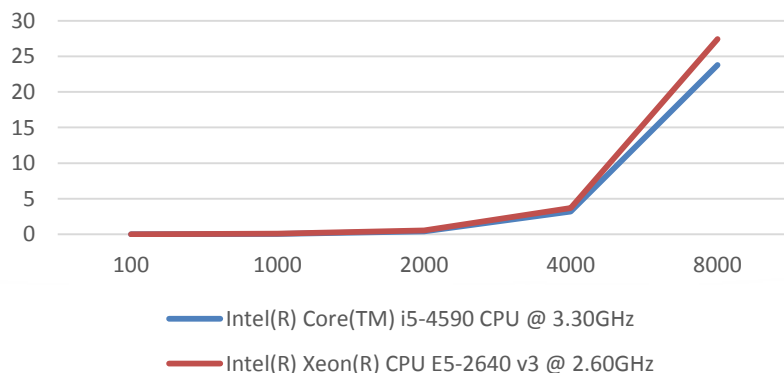
Tiempo (s) matrices 8000 x 8000 $C=A*B$



Gflops por tamaño del problema $C=A*B$



Tiempo (s) con 4 núcleos por tamaño del problema $C=A*B$



Aceleradoras: Xeon Phi

- El nodo 21 cuenta dos tarjetas **Xeon Phi coprocessor 5100 series (rev 11)**
- Cada una cuenta con:
 - 8 Gb de memoria GDDR5 (320 GB/s)
 - 60 “cores” con 4 hilos cada uno a **1.05 GHz** (480 hilos en total)
- Cada tarjeta funciona como un servidor independiente al que se accede a través de un interfaz de red (ssh).
- Intel proporciona modelos de programación para la ejecución de programas en interior de las tarjetas (native – offload – cilk_offload) <https://software.intel.com/en-us/articles/native-and-offload-programming-models>

Aceleradoras: NVidia TESLA

- El nodo 22 cuenta con 2 tarjetas **NVIDIA GK110GL [Tesla K20m]**
- Son tarjetas gráficas destinadas a la computación
- Cada una cuenta con 2496 núcleos “CUDA”, funcionando a 706 MHz
- Cuentan con 5 Gb de memoria GDDR5 y una tasa de transferencia de 208 GB/s
- Proporciona un entorno de desarrollo especial, basado en CUDA, así como depurador, análisis de rendimiento, etc.
- <http://www.nvidia.es/object/tesla-high-performance-computing-es.html>

La red Infiniband

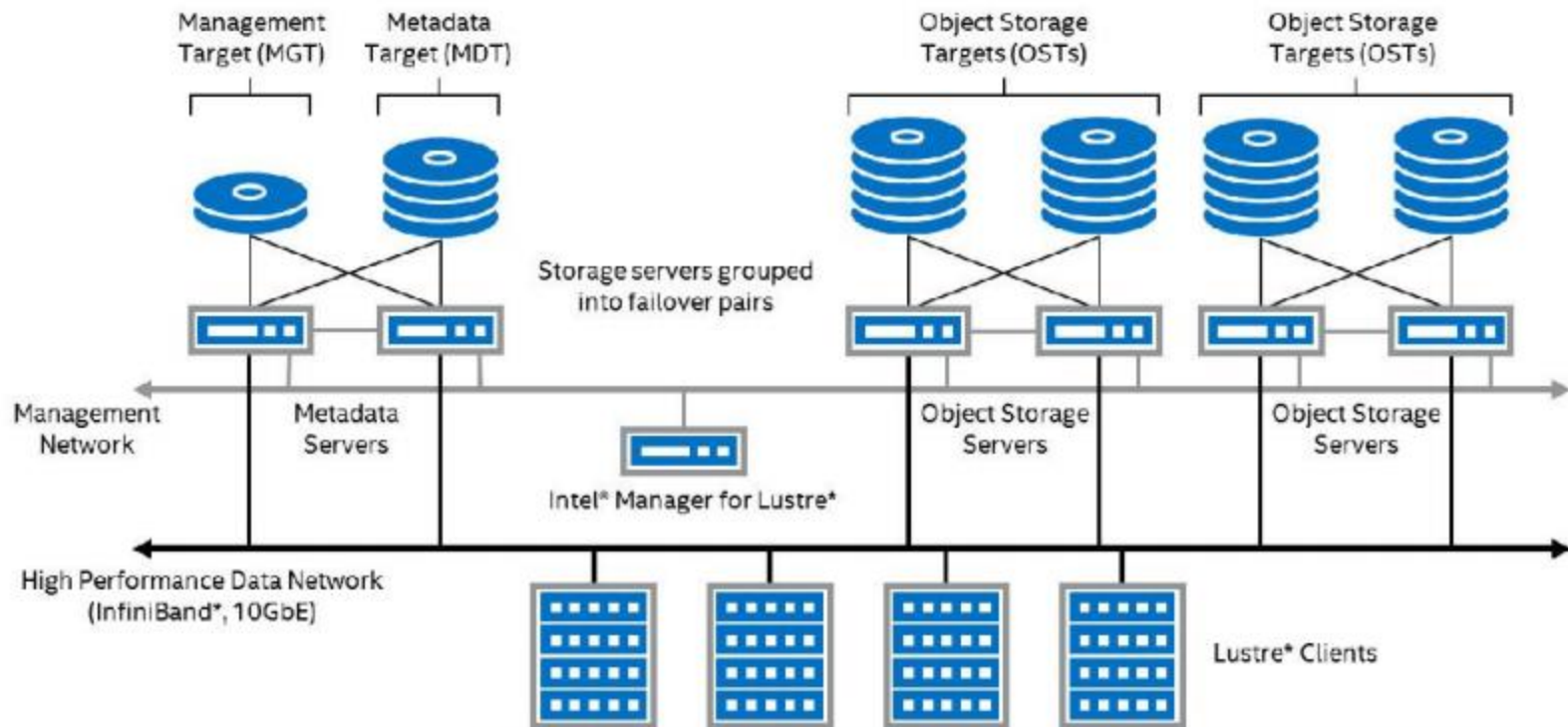
- La red Infiniband conecta los nodos entre sí y con el sistema de almacenamiento (Lustre).
- Infiniband es una red de **alta velocidad** y **baja latencia**, junto con baja sobrecarga de CPU y bajo consumo eléctrico, destinada a conectar los nodos entre sí.
- En el caso del cluster Lovelace, se emplea Infiniband QDR, que alcanza velocidades de hasta **40Gbps**.
- Los nodos tienen configurado además el protocolo IP sobre el medio físico, para una mayor facilidad de empleo por las aplicaciones (IPolb).
- Los desarrollos basados sobre **OpenMPI** utilizarán la infraestructura Infiniband para la transmisión de datos entre procesos.

El sistema de archivos LUSTRE

- **Lustre** es una tecnología de Intel para proporcionar un sistema de ficheros distribuido, escalable, robusto y rápido para sistemas en CLUSTER.
- Especializado en alta disponibilidad y concurrencia.
- Permite lecturas y escrituras simultáneas sobre el mismo fichero, de varios procesos desde diversos servidores.
- El cliente de LUSTRE es compatible POSIX y ofrece una visión transparente de las operaciones sobre ficheros y FILESYSTEMS, además de un conjunto de operaciones exclusivas.
- Las operaciones de **datos** y **metadatos** están separadas, y distintos servidores se encargan de ellas.

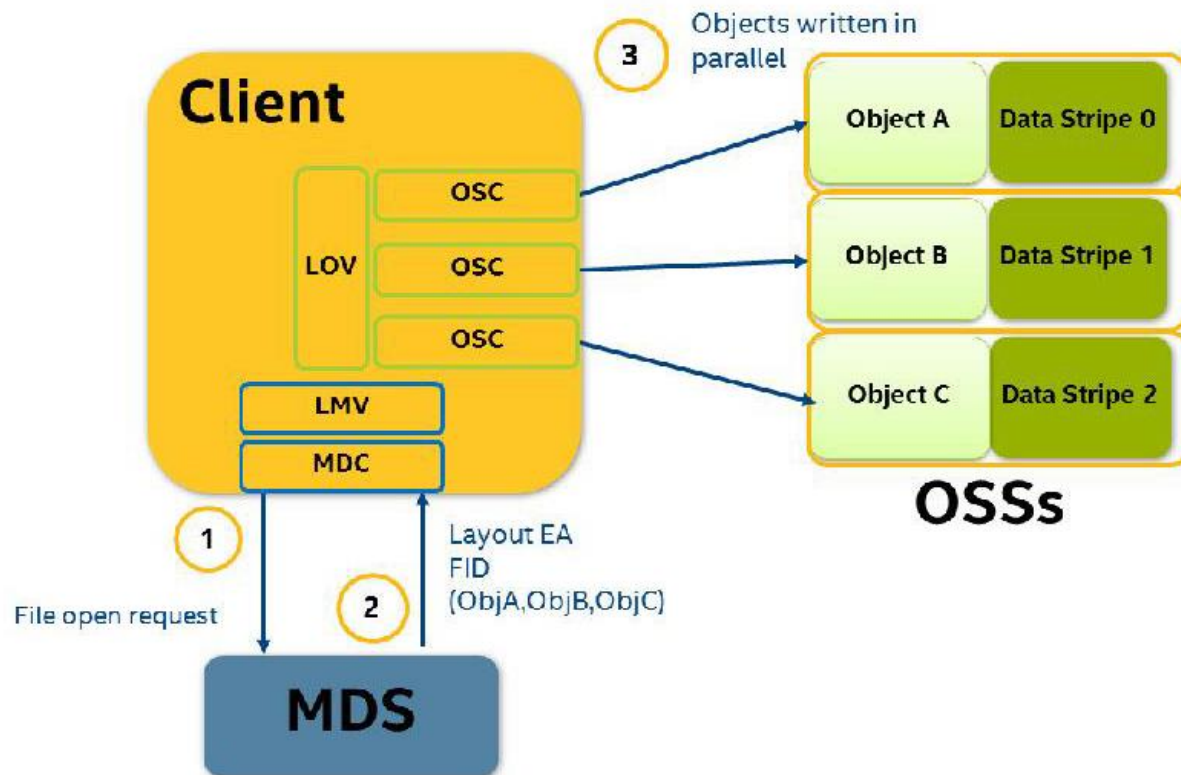
Arquitectura de Lustre

Generic Lustre* File System



Capacidades de Lustre

Basic Lustre* I/O Operation



Lustre: comandos

- `$ ls` es el comando básico para las operaciones específicas sobre el sistema de ficheros.
- Presenta un interfaz CLI con varias opciones
- Comandos básicos: `mkdir`, `find`, `df`, `cp`, `ls`, `mv`
- Comandos intermedios: `quota`, `setquota`, `getstripe`, `setstripe`, `setfacl`, `getfacl`

Arquitectura lógica: SGE

- El gestor de recursos/planificador del cluster es Sun Grid Engine/Open Grid Scheduler
- Desarrollado inicialmente por Sun Microsystems y liberado el código a la comunidad tras la compra por parte de Oracle.
- <http://gridscheduler.sourceforge.net/>
- Otros gestores: Torque, PBS, SLURM, ...
- Se encarga de gestionar los recursos del cluster y planificar la ejecución de los trabajos en los distintos nodos que lo componen.
- Para ello necesita la información que le proporcionamos al lanzar los trabajos.

Arquitectura lógica: SGE

- El nodo “**master**” es el encargado de la planificación, monitorización y lanzamiento de los trabajos. También es el nodo donde hacemos login.
- El resto de nodos son nodos de ejecución de trabajos (estructura “plana”).
- Los comandos de gestión del CLUSTER solo funcionan en nodo “master”.
- Desde este nodo podemos:
 - Editar, preparar y compilar programas
 - Revisar el estado de ejecución de nuestros trabajos
 - Lanzar sesiones interactivas, trabajos batch en serie y trabajos en paralelo.

El sistema de colas

- El software de gestión del cluster permite la creación de colas para el proceso de trabajos.
- Las colas se utilizan para la planificación de trabajos, según las necesidades.
- Podemos consultar las colas definidas en el sistema con **“qconf -sql”**
 - All.q : Cola general para batch, interactivo y paralelo. Corre en todos los nodos excepto los nodos especiales (gpu.q y phi.q)
 - Bigmem.q : Cola para los trabajos que requieren de mucha memoria (> 20GB)
 - Gpu.q : Cola para los trabajos que requieren ejecución en una de las tarjetas NVIDIA tesla.
 - Phi.q : Cola para los trabajos que requieren ejecución en una de las tarjetas Xeon Phi.

Mostrar los hosts y el estado del cluster

- Comando “**qhost**”
- Permite también ver el estado de los trabajos en cada nodo del cluster:
 - \$ qhost -j [-h <nodo>]
- También las colas que atiende cada nodo:
 - \$ qhost -q [-h <nodo>]
- Muestra el número de CPUs (“slots”), la carga total del sistema, la memoria total disponible, la memoria en uso, el área de intercambio total y el área de intercambio en uso.

Envío de trabajos al cluster

- Los tipos de trabajos que podemos lanzar son: sesiones interactivas, trabajos en serie y trabajos en paralelo.
- Todos los trabajos se inician con un comando (qrsh/qlogin y qsub), y se lanzan sobre una de las colas definidas.
- La cola definida por omisión es **all.q**.

Iniciar sesiones interactivas

- Para iniciar las sesiones interactivas se puede utilizar el comando qsh, qrsh o qlogin.
- **qsh** : Inicia una sesión interactiva mediante xwindow
- **qrsh** : Inicia una sesión interactiva mediante “remote shell”
- **qlogin** : Inicia una sesión interactiva mediante “telnet”

Trabajos interactivos

- Permiten trabajar con una Shell, en una máquina que esté menos ocupada dentro de las máquina que sirven esa cola.
- Los accesos desde cualquier máquina al directorio compartido (\$HOME) son idénticos.
- Los nodos comparten la misma configuración
- El trabajo interactivo está limitado a 5 min de tiempo de CPU por proceso
- Para más de 5 min de CPU, debe utilizarse el trabajo BATCH (en serie o paralelo)
- Este tiempo es suficiente para la mayoría de trabajos interactivos, compilación, edición, comandos de shell, etc.

Trabajos batch (en serie)

- Los trabajos en batch se lanzan desde el nodo maestro (master) con el comando
 - `$ qsub [opciones] trabajo`
- Trabajo puede ser un Shell script o un archivo binario ejecutable (opción `-b y`)
- En caso de ser un Shell script puede incluir las opciones de lanzamiento en forma de comentarios con la sintaxis “`#$ opción`”
- Las opciones pueden encontrarse en la página del manual dentro de la propia máquina:
 - `$ man qsub`
- También existe una guía en <https://www.icmat.es/computing/docs>

Ejemplo de script de trabajo en batch (serie)

```
-----mandel_seq.sh-----
#!/bin/bash
#$ -cwd
#$ -q phi.q
#$ -l h_vmem=1G
#$ -N mandel_seq_hd
#$ -j
# Avoid appending information over an existing file
truncate -s 0 $SGE_STDOUT_PATH
# Load runtime environment on remote execution machine
module load mpi/openmpi
module load compilers/gcc/6.3.0
# Ejecuting program with parameters
./Release/mandel_seq -w 3840 -t 2160
-----
[master] $ qsub mandel_seq.sh
```

Monitorizar los trabajos: qstat, qacct

- qstat es la herramienta de línea de comandos para la visualización del estado de ejecución de los trabajos
- qacct puede utilizarse para la visualización de los trabajos ya finalizados.

Importante: especificar los recursos necesarios al planificador

- Cuanto mejor especifiquemos los recursos necesarios al planificador, más posibilidades tendremos de una ejecución correcta y óptima.
- Los recursos necesarios para la ejecución se especifican con el parámetro “-l”, seguido de una lista de recursos necesarios.
- Para ver todos los recursos definidos, puede utilizarse el comando “qhost -F -h <nodo>
- Los recursos pueden ser consumibles o no (se agotan según se utilizan)
 - Se consumen: el espacio en disco, el ancho de banda, la memoria física, la memoria virtual
 - No se consumen: el número total de núcleos, la arquitectura física, la conexión de determinados dispositivos, etc.

Recursos más importantes a especificar

- Memoria virtual:
 - -l h_vmem=10G
- Memoria física:
 - -l mem_free=10G
- Slots (hilos de ejecución)
 - -l slots=8

Trabajos en paralelo

- Los trabajos en paralelo se caracterizan por lanzar un número de procesos que se coordinan entre sí para producir un resultado.
- El paralelismo puede ser:
 - **de grano fino** (los procesos necesitan comunicarse frecuentemente entre sí),
 - **de grano grueso** (necesitan comunicarse, pero con poca frecuencia) o
 - **“embarazosamente paralelos”**, en cuyo caso necesitan comunicarse rara vez o ninguna en absoluto.
- Para cada tipo de problema a paralelizar, se utilizan técnicas diferentes, teniendo en cuenta que la proximidad física de los procesos a transmitir información influye decisivamente en el rendimiento.
 - La transferencia entre núcleos de la misma CPU es más rápida que la transferencia entre núcleos en diferente CPU, pero misma placa.
 - La transferencia entre CPUS en distintas placas (servidores) es mucho más costosa.

Comunicación entre procesos

- A efecto de comunicar procesos diferentes entre sí, se emplean distintas técnicas:
 - Para comunicar procesos dentro del mismo servidor, pueden utilizarse los **mecanismos de comunicación** POSIX: tuberías (pipes), FIFOs, colas de mensajes, memoria compartida, etc.
 - Muy bajo nivel
 - Complicadas de programar
 - Requieren mecanismos de sincronización: regiones críticas, semáforos, etc.
 - **OpenMP** simplifica la programación en C, C++ y FORTRAN para los programas multihilo (pero dentro del mismo servidor)
 - **OpenMPI/MPINCH/Intel MPI**: Implementan una API de MPI (Message Passing Interface) para comunicar procesos que pueden residir o no en el mismo servidor.

Los entornos de ejecución en paralelo

- Para integrar las aplicaciones que utilizan paralelismo con la gestión del cluster, se definen “entornos de paralelismo”.
- Éstos no son más que definiciones de entornos para el lanzamiento de trabajos en paralelo.
- Con “qconf –spl” se pueden visualizar los distintos entornos de ejecución en paralelo.

Ejemplo de script de trabajo en paralelo

```
#!/bin/bash
```

```
#$ -cwd
```

```
#$ -pe mpi 16
```

```
module load mpi/openmpi
```

```
module load compilers/gcc/6.3.0
```

```
mpirun -np 16 mandel_par
```

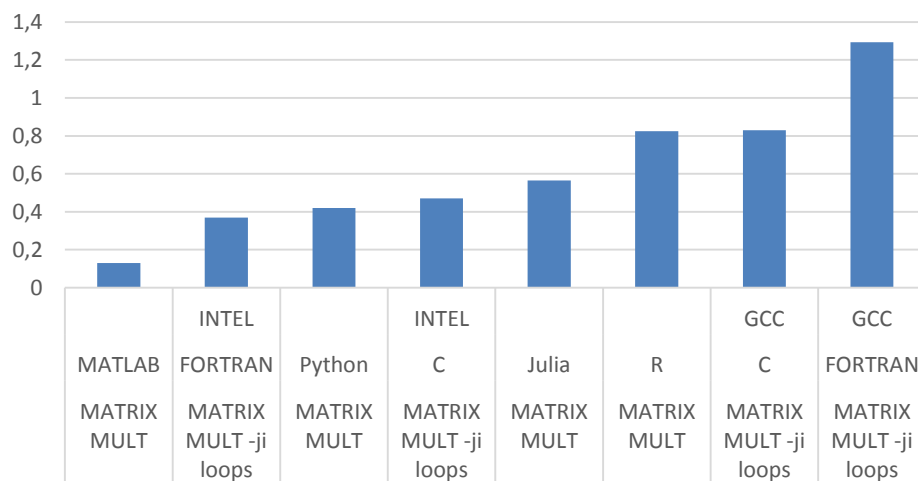
Software y librerías disponibles

- En el cluster está configurada la utilidad de “modules” para las configuraciones de entornos.
- Disponible en la guía de uso de módulos:

<https://www.icmat.es/downloads/computing/hpc-module-files.pdf>

Programación paralela: comparativa lenguajes – multiplicación de matrices

$C = A * B$ (tam 1000x1000x1000)



Problema	LANG	COMP	VERS	OPT	SIZE	TIME(s)
MATRIX MULT	MATLAB		2010b		1000x1000x1000	0,129852
MATRIX MULT -ji loops	FORTTRAN	INTEL	15.0.3	-O3	1000x1000x1000	0,368944
MATRIX MULT	Python		4.4.7		1000x1000x1000	0,42
MATRIX MULT -ji loops	C	INTEL	15.0.3	-O3	1000x1000x1000	0,47
MATRIX MULT	Julia		0.5.1		1000x1000x1000	0,56549118
MATRIX MULT	R		3.3.2		1000x1000x1000	0,825
MATRIX MULT -ji loops	C	GCC	6.5.0	-O3	1000x1000x1000	0,83
MATRIX MULT -ji loops	FORTTRAN	GCC	6.5.0	-O3	1000x1000x1000	1

Programación paralela: ejemplo

- Mandelbrot
- Ejemplo en C++ de programación secuencial y paralela
- En `/LUSTRE/Examples/mandel`

Gracias!!