Organización y gestión de sistemas de altas prestaciones

Ivan Rodero Castro
Francesc Guim Bernat

PID_00218839

UOC
Universitat Oberta de Catalunya

Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0 España de Creative Commons. Podéis copiarlos, distribuirlos y transmitirlos públicamente siempre que citéis el autor y la fuente (FUOC. Fundación para la Universitat Oberta de Catalunya), no hagáis de ellos un uso comercial y ni obra derivada. La licencia completa se puede consultar en http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es

Índice

Introducción ... 5

Objetivos ... 6

1. Organización de los sistemas de altas prestaciones ... 7

2. Redes de interconexión ... 10
- 2.1. Redes de sistemas de memoria compartida ... 10
- 2.2. Redes de sistemas de memoria distribuida ... 12
- 2.3. Latencia y ancho de banda ... 16

3. Sistemas de archivos para sistemas de altas prestaciones ... 17
- 3.1. Sistemas de archivos distribuidos ... 17
  - 3.1.1. Network File System (NFS) ... 18
- 3.2. Sistemas de archivos paralelos ... 20
  - 3.2.1. General Parallel File System (GPFS) ... 21
  - 3.2.2. Linux Cluster File System (LUSTRE) ... 24

4. Sistemas de gestión de colas y planificación ... 29
- 4.1. Sistemas de gestión de colas ... 30
  - 4.1.1. PBS (*portable batch system*) ... 30
- 4.2. Planificación ... 36
  - 4.2.1. Políticas de planificación basadas en *backfilling* ... 39

Bibliografía ... 43

Introducción

En este módulo didáctico estudiaremos la organización y gestión de los sistemas de altas prestaciones. También se presentarán las características de los componentes principales de los sistemas de altas prestaciones, como son la red de interconexión y sistema de archivos de altas prestaciones.

Finalmente, se introducirá el entorno de ejecución de aplicaciones paralelas para computación de altas prestaciones. Estudiaremos los componentes de software utilizados en los sistemas paralelos orientados a las altas prestaciones, así como políticas de planificación, que permiten optimizar la ejecución de las aplicaciones paralelas y la utilización de los sistemas de altas prestaciones.

Objetivos

Los materiales didácticos de este módulo contienen las herramientas necesarias para lograr los objetivos siguientes:

1. Entender el funcionamiento y organización de los sistemas de altas prestaciones.

2. Conocer los fundamentos de los componentes básicos de los sistemas de altas prestaciones.

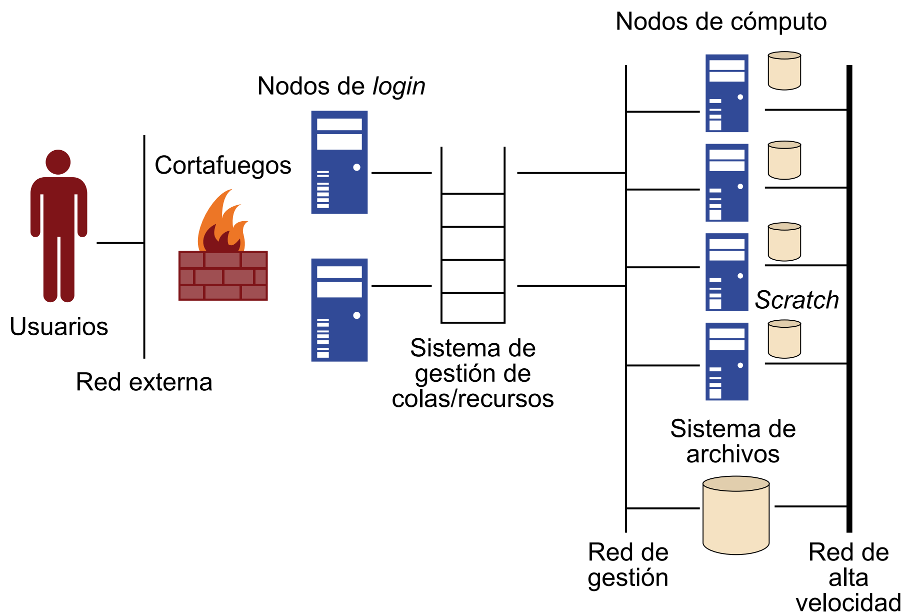
3. Conocer el funcionamiento y los componentes que forman los sistemas de gestión de aplicaciones en sistemas paralelos para computación de altas prestaciones.

4. Aprender los conceptos fundamentales de las políticas de planificación de trabajos en entornos de altas prestaciones.

1. Organización de los sistemas de altas prestaciones

Tal y como vimos en el primer módulo didáctico de esta asignatura, actualmente los sistemas de altas prestaciones o supercomputadores son sistemas de tipo clúster. Anteriormente, los sistemas de memoria compartida y los multiprocesadores habían sido populares, pero la necesidad de más nivel de paralelismo y la aparición de redes de interconexión de altas prestaciones ha hecho que los sistemas de altas prestaciones actuales sean un conjunto de computadores independientes, interconectados entre sí, que trabajan conjuntamente para resolver un problema. Así pues, los computadores y la red de comunicación son elementos esenciales en un sistema de altas prestaciones, pero también hay otros elementos clave, tal como muestra la figura 1.

Figura 1. Elementos básicos de un sistema de altas prestaciones típico.



Los computadores de los sistemas de altas prestaciones se organizan principalmente en computadores (o nodos) de dos tipos diferentes:

- Nodos de *login*: Son los nodos accesibles por los usuarios, habitualmente mediante conexiones seguras de tipo `ssh`. Los supercomputadores normalmente disponen de varios nodos de *login*. Este tipo de nodos facilitan funciones básicas, como por ejemplo, crear y editar ficheros, compilar código fuente, utilizar herramientas para estudiar el rendimiento de aplicaciones, hacer pruebas de funcionalidad, enviar y gestionar trabajos al sistema de colas, etc. En algunos casos los nodos de *login* son del mismo tipo que los nodos de cómputo, pero pueden ser de diferentes tipos e incluso tener una arquitectura diferente. En el último caso, el usuario tiene que utilizar opciones específicas de compilación para que el programa pueda ejecutarse correctamente acorde a los nodos de cómputo. Como parte de

la infraestructura de seguridad del sistema, también se acostumbran a utilizar cortafuegos para aislar el sistema de la red externa.

- Nodos de cómputo: Son los que ejecutan las aplicaciones paralelas utilizando una red de alta velocidad. A pesar de que pueden ser heterogéneos, normalmente son homogéneos o bien tienen diferentes particiones que son homogéneas. Por ejemplo, puede haber un conjunto de nodos con CPU, otros que incluyen GPU, y otros que se utilizan para la visualización y tienen características específicas, como por ejemplo, mayor capacidad de memoria.

Uno de los elementos esenciales de los sistemas de altas prestaciones y que los diferencian de otros sistemas, como por ejemplo los centros de datos, es la red de interconexión. Normalmente se puede encontrar una red de alta velocidad, es decir, de gran ancho de banda pero con latencias muy reducidas, y otra de gestión que no interviene en la ejecución de las aplicaciones paralelas (en el paso de mensajes). Algunos ejemplos de redes de altas prestaciones son Infiniband, Cray Gemini o Myrinet. Las características de los diferentes tipos de redes las estudiaremos más adelante en este módulo didáctico.

Los sistemas de altas prestaciones también acostumbran a utilizar un sistema de archivos paralelo compartido por todos los nodos del sistema. Tal y como veremos más adelante, este tipo de sistema permite mejorar el rendimiento y también otras cuestiones, como por ejemplo la escalabilidad, con un número muy elevado de nodos o la utilización de grandes volúmenes de datos. De forma complementaria, los nodos pueden tener un espacio de almacenamiento temporal (conocido como *scratch*), por ejemplo, en forma de disco o SSD (*solid-state drive*) para mejorar la localidad de los datos cuando no hay que compartirlos. De este modo se puede reducir la utilización de la red de interconexión y del sistema de archivos paralelo, pero hay que tener en cuenta que solo puede hacerse al final de la ejecución, al mover los datos a un soporte de almacenamiento permanente. Finalmente, también se pueden encontrar dispositivos de copia de seguridad o de almacenamiento masivo, como por ejemplo basados en cintas que ofrecen gran capacidad pero con mucha más latencia.

Para poder utilizar los recursos eficientemente y compartirlos entre múltiples usuarios, los sistemas de altas prestaciones usan sistemas de colas como interfaz entre el usuario y los nodos de cómputo. Además de mantener los trabajos de los usuarios del sistema, también los gestiona mediante políticas de planificación, tal como veremos más adelante en este módulo didáctico.

A nivel de software de sistema, hay que destacar que actualmente los sistemas de altas prestaciones acostumbran a utilizar sistemas basados en Unix, especialmente Linux. Además, también incluye *middleware*, que tiene los siguientes objetivos:

- Proporcionar transparencia al usuario de modo que no tenga que preocuparse de los detalles de bajo nivel.

- Escalabilidad del sistema.

- Disponibilidad del sistema para soportar las aplicaciones de los usuarios.

- El concepto de SSI (*single system image*) permite al usuario ver el clúster de forma unificada como si fuera un único recurso o sistema de cómputo.

Desde el punto de vista del entorno de ejecución y desarrollo, el software habitual, como por ejemplo compiladores y paquetes matemáticos, se acostumbra a gestionar a través de módulos. Esto permite modificar el entorno de forma dinámica (por ejemplo, utilizar una cierta versión de un compilador). Algunas acciones útiles cuando se utilizan módulos son consultar los módulos disponibles (`module avail`), cargar un módulo (`module load<aplicación>`) o descargar (`module unload<aplicación>`).

En los próximos capítulos estudiaremos las características básicas de los elementos que forman los sistemas de altas prestaciones y los tipos más comunes que se encuentran en la actualidad.

2. Redes de interconexión

Tal y como se ha comentado, la red de interconexión desempeña un papel decisivo en el rendimiento de los sistemas de memoria distribuida y de los de memoria compartida. Aunque tuvierais procesadores y memorias de un rendimiento ilimitado, una red de interconexión lenta haría que el rendimiento de los programas paralelos se degradara muy considerablemente.

A pesar de que algunas de las redes de interconexión tienen aspectos en común, hay muchas diferencias entre las redes de interconexión para sistemas de memoria compartida y las de sistemas de memoria distribuida.

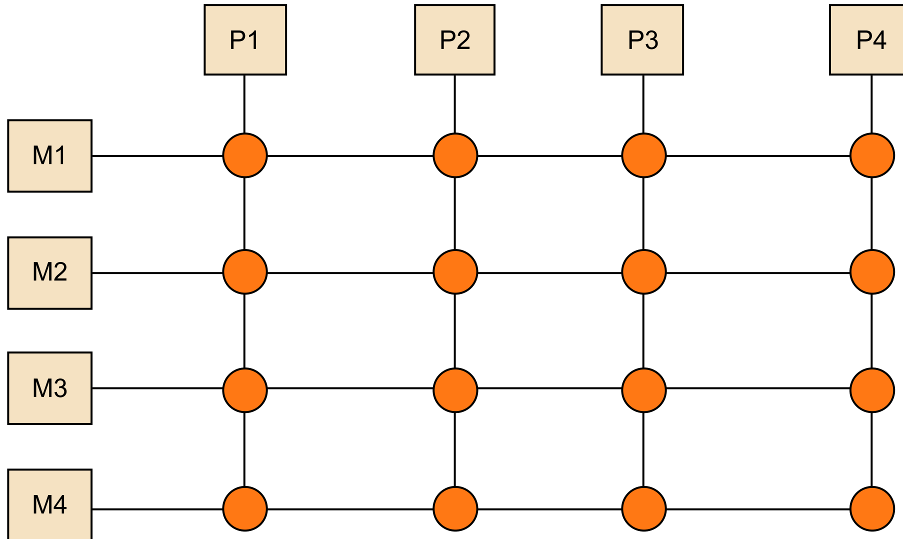
2.1. Redes de sistemas de memoria compartida

Actualmente las dos redes de interconexión más utilizadas en sistemas de memoria compartida son los buses y los *crossbars*. Hay que recordar que un bus no es más que un conjunto de cables de comunicación en paralelo junto con hardware que controla el acceso al bus. La característica clave de un bus es que los cables de comunicación son compartidos por todos los dispositivos que están conectados. Los buses tienen la ventaja de tener un cuerpo reducido y flexibilidad, y múltiples dispositivos se pueden conectar al bus con muy poco coste adicional. Aun así, como los cables de comunicación son compartidos, a medida que el número de dispositivos que utilizan el bus crece, aumentan las posibilidades de que haya contención por la utilización del bus y, en consecuencia, su rendimiento se reduce. Por lo tanto, si conectamos un número elevado de procesadores a un bus, hemos de suponer que estos deberán esperarse frecuentemente para acceder a la memoria principal. Así pues, a medida que el tamaño del sistema de memoria compartida aumenta, los buses son reemplazados por redes de interconexión *switched*.

Las redes de interconexión *switched* utilizan interruptores[1] para controlar el enrutamiento de los datos entre los dispositivos que hay conectados. La figura siguiente muestra un *crossbar*, donde las líneas son enlaces de comunicación bidireccional, los cuadrados son núcleos o módulos de memoria, y los círculos son interruptores.

(1)En inglés, *switches*.

Figura 2. *Crossbar* conectando 4 procesadores y 4 memorias



Los interruptores individuales pueden tener una o dos configuraciones, tal como muestra la figura 3. Con estos interruptores y al menos tantos módulos de memoria como procesadores, solo habrá conflictos entre dos núcleos intentando acceder a la memoria si los dos núcleos intentan hacerlo simultáneamente al mismo módulo de memoria. Por ejemplo, la figura 4 muestra la configuración de los interruptores si P1 escribe a M4, P2 lee de M3 , P3 lee de M1 y P4 escribe a M2.

Figura 3. Configuración de los interruptores en un crossbar

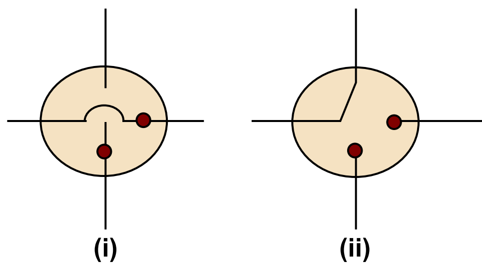
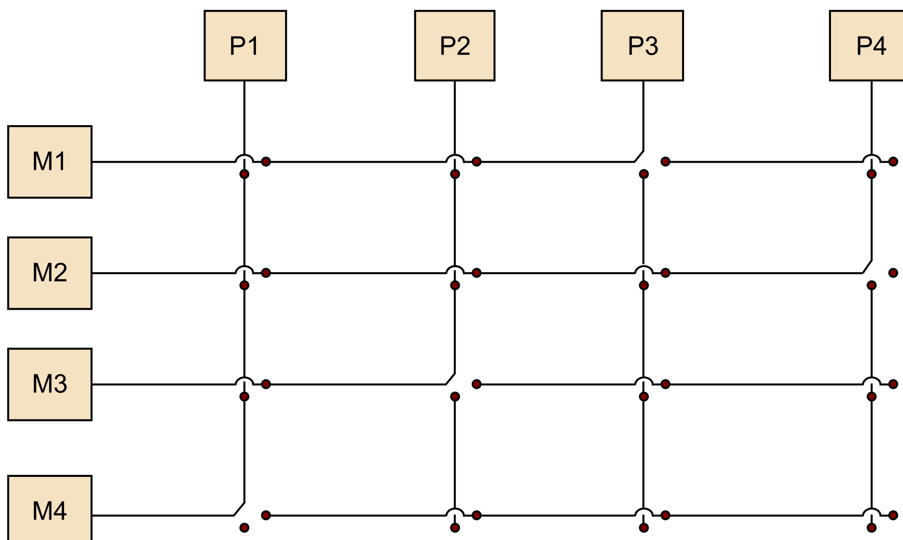


Figura 4. Acceso a memoria por varios procesadores simultáneamente



Los *crossbars* permiten la comunicación simultánea entre los diferentes dispositivos, por lo tanto son mucho más rápidos que los buses. En cambio, el coste de los interruptores y de los enlaces es relativamente alto. En general, un sistema tipo bus es bastante más barato que uno basado en *crossbar* para un sistema pequeño.

2.2. Redes de sistemas de memoria distribuida

Las redes de interconexión de sistemas de memoria distribuida normalmente se dividen en dos grupos: interconexiones directas e interconexiones indirectas. En una interconexión directa cada uno de los interruptores se conecta a un par procesador-memoria y los interruptores están interconectados entre ellos. Las figuras 5 y 6 muestran un anillo y una malla toroidal[2] de dos dimensiones, respectivamente. Como anteriormente, los círculos son interruptores, los cuadrados son procesadores y las líneas son enlaces bidireccionales. Un anillo es mejor que un simple bus, puesto que permite múltiples conexiones simultáneas. En cambio, se puede ver claramente que habrá situaciones en las que algunos procesadores tendrán que esperar a que otros acaben sus comunicaciones. La malla toroidal es más cara que un anillo porque los interruptores son más complejos (han de tener cinco enlaces en lugar de los tres del anillo) y si hay p procesadores, el número de enlaces es $3p$ en una malla toroidal, mientras que en un anillo solo sería de $2p$.

(2)En inglés, *toroidal mesh*.

Figura 5. Anillo

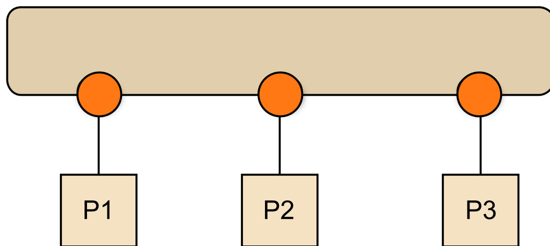
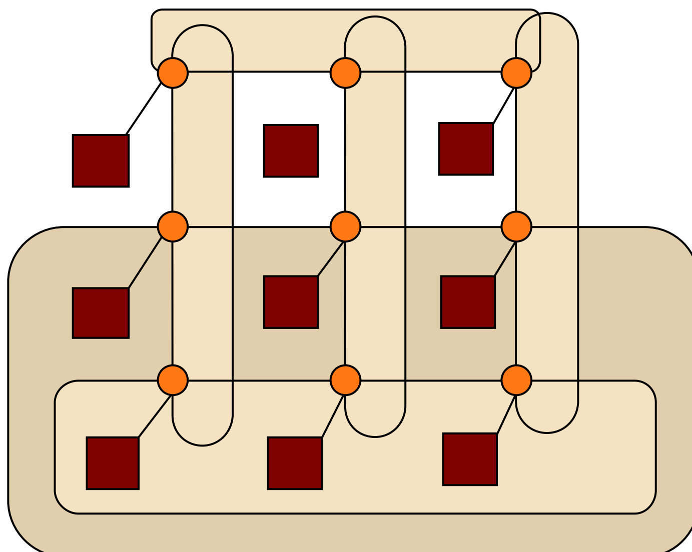


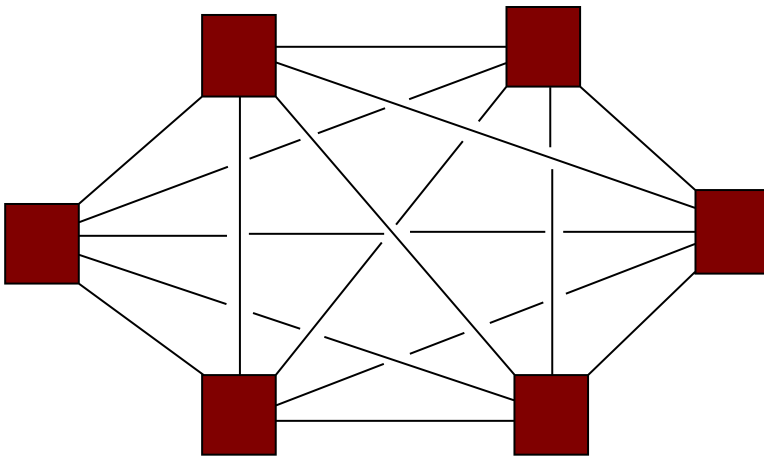
Figura 6. Malla toroidal



El ancho de banda[3] de un enlace es la ratio a la que puede transmitir datos. Normalmente se da en megabits o megabytes por segundo. La red de interconexión directa ideal es la completamente conectada, en la que cada interruptor está conectado a todos los otros, tal como muestra la figura siguiente. El problema de este tipo de red es que no se pueden construir para sistemas de más de unos pocos nodos, puesto que requieren un total de $p^2/p + p/2$ enlaces, y cada interruptor se debe poder conectar a p enlaces. Así pues, se trata más bien de una red de interconexión teórica en el mejor de los casos que de una práctica, y se utiliza como referencia para la evaluación de otros tipos de red de interconexión.

(3)En inglés, *bandwidth*.

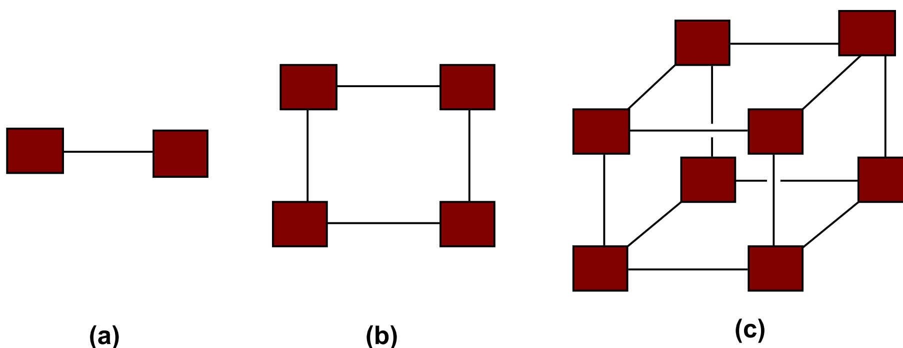
Figura 7. Red completamente conectada



Un hipercubo[4] es una red de conexión directa altamente conectada que se ha utilizado en sistemas actuales. Los hipercubos se construyen de forma inductiva: un hipercubo de una dimensión es un sistema completamente conectado con dos procesadores. Un hipercubo de dos dimensiones se construye a partir del de una dimensión, uniendo los correspondientes interruptores, y uno de tres dimensiones se hace del mismo modo a partir de uno de dos dimensiones, tal como muestra la figura siguiente. Por lo tanto, un hipercubo de dimensión d tiene $p = 2\ d$ nodos, y un interruptor en un hipercubo de d dimensiones está conectado directamente a un procesador y d interruptores. Así pues, un hipercubo con p nodos es más costoso de construir que una malla toroidal.

(4)En inglés, *hypercube*.

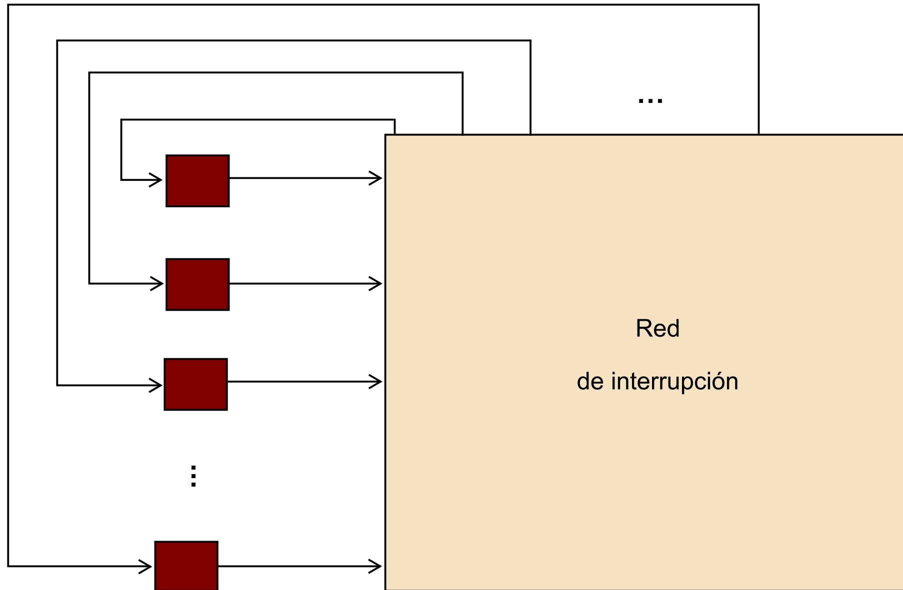
Figura 8. Hipercubos de (a) una, (b) dos y (c) tres dimensiones



Las interconexiones indirectas son una alternativa a las conexiones directas donde los interruptores pueden no estar conectados directamente a un procesador. Muchas veces son vistos como enlaces unidireccionales y un conjunto de procesadores, cada uno de los cuales tiene un enlace de entrada y uno de salida, y una red de interrupción[5], tal como muestra la figura 9.

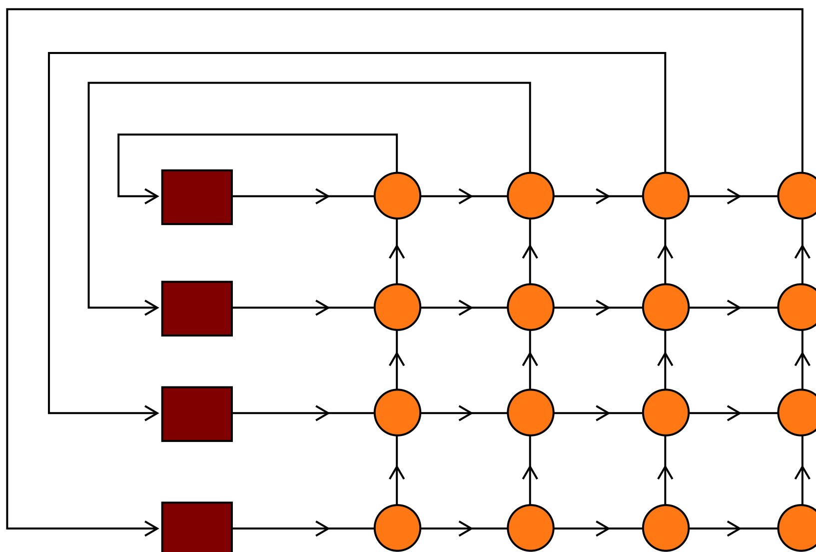
(5)En inglés, *switching network*.

Figura 9. Red de interrupción genérica



El *crossbar* y la red omega son ejemplos relativamente simples de redes de interconexión indirectas. La figura 10 muestra el diagrama de un *crossbar* de memoria distribuida, que tiene enlaces unidireccionales, a diferencia del *crossbar* de memoria compartida, que tiene enlaces bidireccionales, tal como vimos en la figura 4. Hay que tener en cuenta que mientras que dos procesadores no intenten comunicarse con el mismo procesador, todos los procesadores pueden comunicarse simultáneamente con otro procesador.

Figura 10. Interconexión con *crossbar* de memoria distribuida



En una red omega como la de la figura 11, los interruptores son *crossbars* de dos-a-dos, tal como muestra la figura 12. Hay que observar que al contrario del *crossbar*, hay comunicaciones que no pueden producirse simultáneamente. Por ejemplo, en la figura 11 si el procesador 0 manda un mensaje al procesador 6, entonces el procesador 1 no puede mandar un mensaje al procesador 7 simultáneamente. Por otro lado, la red omega es menos costosa que el *crossbar*, puesto que la red omega utiliza $\frac{1}{2}p \cdot \log_2(p)$ de los interruptores de los *crossbars* 2×2 y, por lo tanto, utiliza un total de $2p \cdot \log_2(p)$ interruptores, mientras que el *crossbar* utiliza p^2.

Figura 11. Red omega

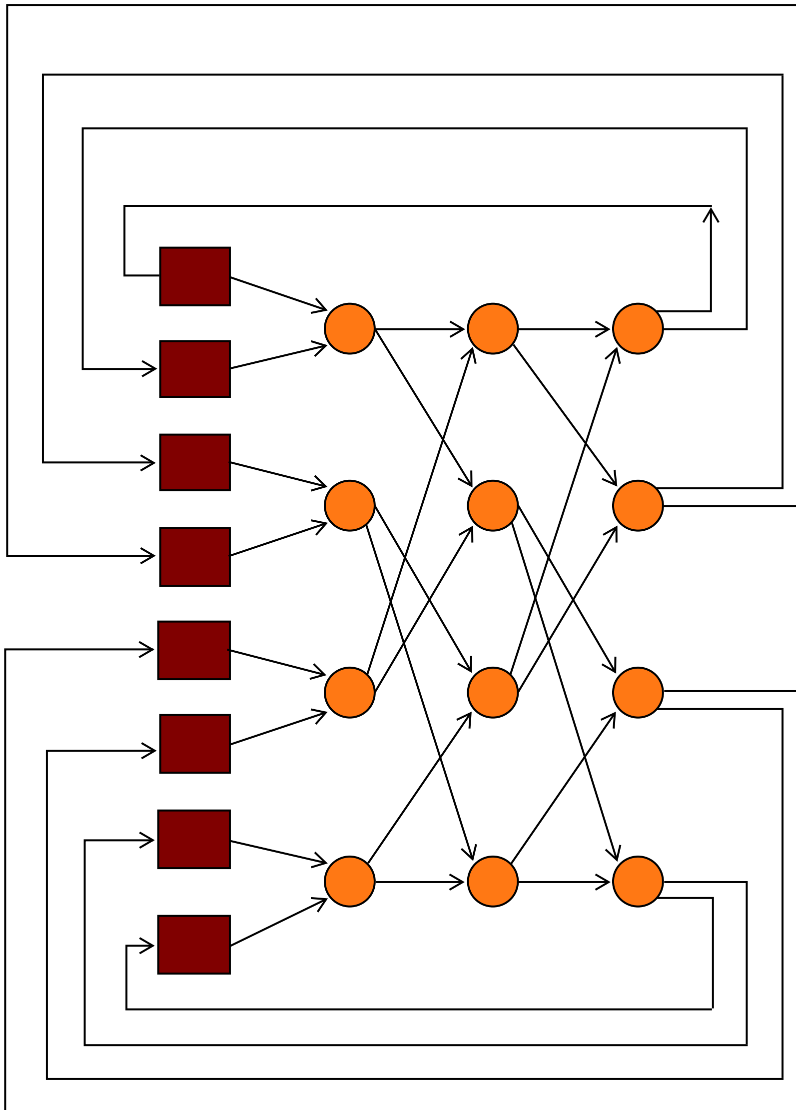
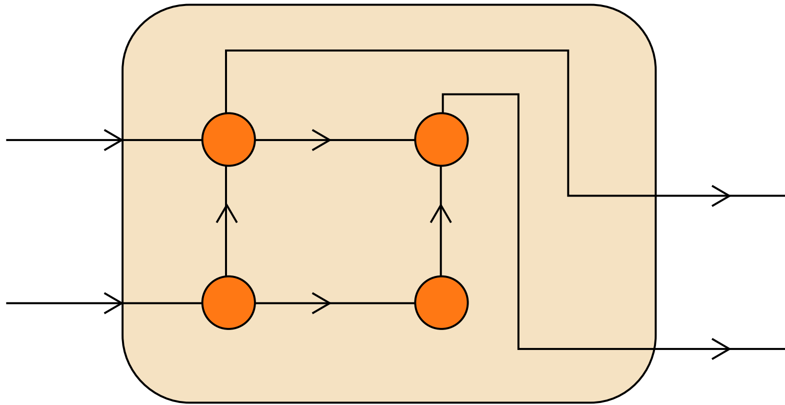


Figura 12. Interruptor de una red omega



2.3. Latencia y ancho de banda

Siempre que hay datos que se transmiten estamos interesados en saber cuánto tardarán los datos en llegar a su destino. Esto es así si hablamos de transmitir datos entre la memoria principal y la memoria caché, memoria caché y registro, disco duro y memoria, o entre dos nodos en un sistema de memoria distribuida o uno híbrido. Hay dos términos que se suelen utilizar para describir el rendimiento de una red de interconexión independientemente de lo que conecte: la latencia y el ancho de banda.

> La **latencia** es el tiempo que pasa entre que el origen empieza a transmitir los datos y el destino empieza a recibir el primer byte. El **ancho de banda** es la ratio donde el destino recibe datos después de que haya empezado a recibir el primer byte.

Por lo tanto, si la latencia de una red de interconexión es l segundos y el ancho de banda es b bytes por segundo, entonces el tiempo que tardaría en transmitir un mensaje de n bytes sería:

```
Tiempo de transmisión de un mensaje = l + n/b
```

3. Sistemas de archivos para sistemas de altas prestaciones

En este capítulo se presentan brevemente las características y cuestiones básicas de los sistemas de archivos que se utilizan en sistemas de altas prestaciones. También se ilustran algunos casos representativos de este tipo de sistemas de archivos, empezando por NFS como fundamento de los sistemas de archivos distribuidos, y continuando con sistemas de archivos paralelos enfocados a las altas prestaciones.

3.1. Sistemas de archivos distribuidos

En general, un sistema de archivos distribuido permite a los procesos el acceso transparente y eficiente de archivos que permanecen en servidores remotos. Las principales tareas son las de organización, almacenamiento, recuperación, compartimento y protección de los archivos. También proporcionan una interfaz de programación, que oculta a los programadores los detalles de localización y cómo se realiza realmente el almacenamiento.

Algunas de las ventajas de estos sistemas de archivos son, entre otras:

- Un usuario puede acceder a sus mismos archivos desde diferentes máquinas.

- Diferentes usuarios pueden acceder a los mismos archivos desde diferentes máquinas.

- Es más fácil de administrar puesto que solo hay un servidor o grupo de servidores.

- Se mejora la fiabilidad puesto que se puede añadir, por ejemplo, tecnología RAID (*redundant array of independent disks*).

A la vez, hay algunos retos y cuestiones importantes que estos tipos de sistemas de archivos intentan solucionar. Por ejemplo:

- Escalabilidad: El servicio se tiene que poder extender incrementalmente para gestionar un amplio rango de cargas y medidas de red.

- Tolerancia a fallos: Clientes y servidores tienen que operar correctamente ante fallos.

- Consistencia: Diferentes clientes tienen que ver el mismo directorio y contenido de los archivos si acceden al mismo tiempo.

- Seguridad: Mecanismos de control de acceso y autenticación.

- Eficiencia: Su utilización tiene que ser similar a la de los sistemas de ficheros locales.

3.1.1. Network File System (NFS)

NFS es un sistema de compartición de archivos entre máquinas de una red de tal manera que tenemos la impresión de trabajar en nuestro disco duro local. Un sistema Linux puede trabajar tanto como servidor o como cliente de NFS (o ambos), lo que implica que puede exportar sistemas de archivos a otros sistemas, así como montar los sistemas de archivos que otras máquinas exportan.

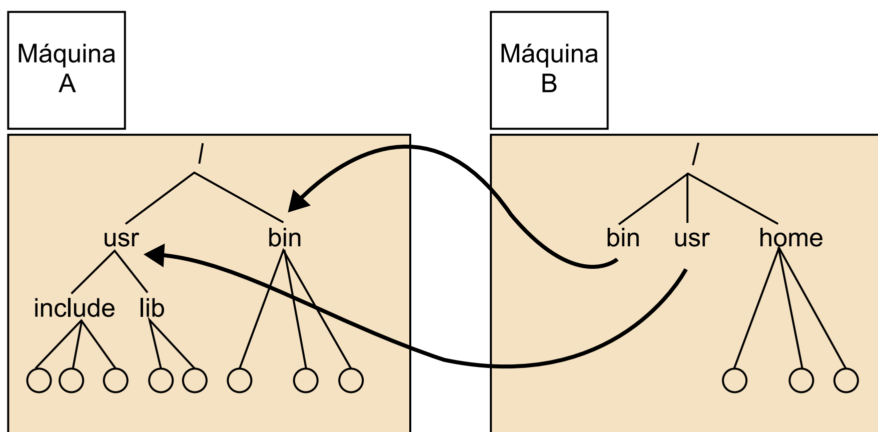
Para soportar NFS, Linux utiliza una combinación de apoyo a nivel de núcleo y demonios en continua ejecución para proporcionar la compartición de archivos NFS. Así pues, el soporte NFS tiene que estar activo en el núcleo del sistema operativo. Por otro lado, NFS utiliza RPC (*remote procedure calls*) para encaminar peticiones entre clientes y servidor. El uso de RPC implica que el servicio *portmap* tiene que estar disponible y activo. El *portmap* es un demonio encargado de crear conexiones RPC con el servidor y de permitir o no el acceso.

Nota

Estudiaremos RPC en el módulo didáctico enfocado a sistemas distribuidos por altas prestaciones.

La figura 13 muestra un ejemplo de utilización de NFS mediante el montaje de directorios remotos para poder compartir archivos.

Figura 13. Ejemplo de utilización de NFS para compartir archivos mediante el montaje de directorios remotos.



En el ejemplo, la máquina A exporta los directorios /usr y /bin, y en la máquina B se montan para su utilización.

Los principales procesos RPC necesarios son los siguientes, a pesar de que intervienen más:

- *rpc.mountd*: Es un proceso que recibe la petición de montaje desde un cliente NFS y comprueba si coincide con un sistema de archivos actualmente exportado.

- *rpc.nfsd*: Es un proceso que implementa los componentes del espacio del usuario del servicio NFS. Trabaja con el núcleo Linux para satisfacer las demandas dinámicas de clientes NFS, como por ejemplo, proporcionar procesos adicionales del servidor para que los clientes NFS lo utilicen.

La configuración del servidor NFS, que se realiza mediante el fichero /etc/exports, permite especificar cuestiones muy importantes relacionadas con la compartición de archivos. Entre estas, se encuentra la lista de máquinas autorizadas en la compartición junto con ciertas opciones, como por ejemplo:

- *ro*: Solo lectura (*read-only*). Las máquinas no pueden cambiar el sistema de ficheros.

- *rw*: Lectura-escritura (*read-write*).

- *async*: Permite al servidor escribir los datos en el disco cuando lo crea conveniente.

- *sync*: Todas las escrituras en el disco hay que hacerlas antes de devolver el control al cliente.

- *wdelay*: El servidor NFS retrasa la escritura en disco cuando hay sospecha de que otra petición de escritura es inminente.

- *no_wdelay*: Para desactivar la opción anterior, la cual solo funciona si se usa la opción sync.

- *root_squash*: Proporciona a los usuarios privilegiados (*root*) conectados remotamente tener ciertos privilegios, como *root* local.

- *no_root_squash*: Para desactivar la acción anterior.

- *all_squash*: Para reconvertir a todos los usuarios.

También se pueden especificar grupos de redes mediante grupos de nombres de dominio o direcciones IP.

Actividad

Buscad información del funcionamiento interno de NFS y pensad cómo realizaríais vuestra propia implementación de un servidor/cliente de NFS.

3.2. Sistemas de archivos paralelos

Los sistemas de archivos paralelos intentan dar solución a la necesidad de entrada/salida de ciertas aplicaciones que gestionan volúmenes masivos de datos y, por lo tanto, requieren un número enorme de operaciones de entrada/salida sobre los dispositivos de almacenamiento. De hecho, en los últimos años se ha visto un crecimiento muy significativo de capacidad de los discos, en cambio, no ha sucedido lo mismo en sus prestaciones, como por ejemplo, ancho de banda y latencia. Esto ha producido un desequilibrio importante entre capacidad de procesamiento y entrada/salida, que hace que afecte muy negativamente a las aplicaciones que son muy intensivas en operaciones de entrada/salida.

Los sistemas de archivos paralelos se basan en el mismo principio que el cómputo para mejorar las prestaciones, es decir, proporcionar entrada/salida paralela. Esto hace que la distribución de datos sea entre múltiples dispositivos de almacenamiento y que se pueda acceder a los datos en paralelo. Hay diferentes tipos de conexión de dispositivos de almacenamiento:

- DAS (*direct-attached storage*): Solución "clásica" de disco asociado a nodo.

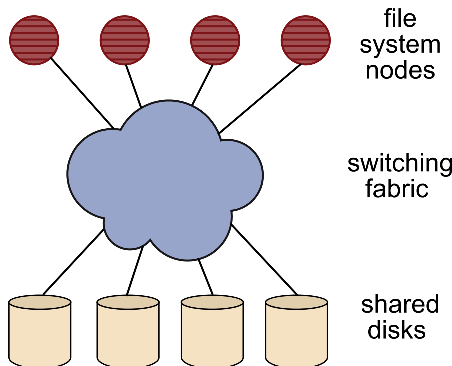
- NAS (*network-attached storage*): Nodo que gestiona un conjunto de discos.

- SAN (*storage area networks*): Red dedicada al almacenamiento.
  - Almacenamiento no vinculado a ningún nodo ("discos de red").
  - Redes de comunicación separadas para datos de aplicación y ficheros.
  - Redes de almacenamiento incluyen *hubs*, *switches*, etc. La tecnología más utilizada es Fibre Channel.
  - Conectividad total entre nodos y dispositivos: cualquier nodo puede acceder directamente a cualquier dispositivo.
  - Conectividad directa entre dispositivos: por ejemplo, para hacer copias entre dispositivos (agiliza copias de seguridad, replicación, etc.).

Respecto a la organización de los sistemas de archivos paralelos, estos usan técnicas de *stripping*, que consisten en almacenar los datos de archivo distribuidos entre discos del sistema de forma similar al RAID-0, pero para software y entre varios nodos. Los sistemas de archivos se comparten a partir de un reparto funcional en dos niveles. El nivel inferior es el servicio de almacenamiento distribuido (por ejemplo, SAN) y el superior es el sistema de archivos en cada nodo de cómputo. Para acceder a los discos como si fueran locales,

cada nodo de cómputo tiene que gestionar metainformación de los datos. La figura 14 ilustra la organización conceptual en capas de los sistemas de archivos paralelos.

Figura 14. Organización funcional por capas de un sistema de archivos paralelo



Algunos ejemplos de sistemas de archivos paralelos son GPFS, Lustre, PVFS o Google File System. A continuación se describen las características básicas de los dos primeros a modo de ejemplo.

3.2.1. General Parallel File System (GPFS)

GPFS fue desarrollado por IBM, y como sistema de archivos paralelo, permite a los usuarios compartir el acceso a datos que están dispersos en múltiples nodos, así como interacción a través de las interfaces estándar de UNIX.

GPFS permite mejorar el rendimiento del sistema, y sus principales características son:

- Garantiza la consistencia de los datos.
- Tiene una alta recuperabilidad y disponibilidad de los datos.
- Proporciona una alta flexibilidad al sistema.
- Administración simplificada.

La mejora de rendimiento es debida a factores como los siguientes:

- Permite que múltiples procesos o aplicaciones accedan simultáneamente a los datos desde todos los nodos utilizando llamamientos estándar del sistema.

- Incremento del ancho de banda de cada uno de los nodos que intervienen en el sistema GPFS.

- Balancea la carga uniformemente entre todos los nodos del sistema GPFS. Un disco no puede tener más actividad que otro.

- Soporta datos de grandes dimensiones.

- Permite lecturas y escrituras concurrentes desde cualquier nodo del sistema GPFS.

GPFS utiliza un sofisticado sistema de administración que garantiza la consistencia de los datos a la vez que permite múltiples e independientes rutas para archivos con el mismo nombre desde cualquier lugar del clúster. Cuando la carga de ciertos nodos es muy alta, GPFS puede encontrar una ruta alternativa por el sistema de archivos de datos.

Para recuperar y facilitar la disponibilidad de los datos, GPFS crea registros *logs* separados para cada uno de los nodos que intervienen en el sistema. GPFS permite que se organice el hardware dentro de un número de grupo de falla. Además, la función de replicación de GPFS permite determinar cuántas copias de los archivos hay que mantener. GPFS también permite añadir recursos, ya sean discos o nodos, dinámicamente, sin necesidad de parar y volver a poner en marcha el sistema.

En un sistema GPFS, cada uno de sus nodos se organiza a partir de los siguientes componentes:

- Comandos de gestión/administración.
- Extensiones del núcleo.
- Un demonio multi-hilo.
- Una capa portable de código abierto.

La extensión del núcleo proporciona una interfaz entre el sistema operativo y el sistema GPFS, lo que facilita la manipulación de los datos en un entorno GPFS, puesto que con los comandos del sistema operativo se puede realizar cualquier operación sobre el sistema GPFS. Por ejemplo, para copiar un archivo solo hay que ejecutar la sintaxis del comando habitual cp fitxer.txt prova.txt.

Los demonios del GPFS se ejecutan en todos los nodos de entrada/salida y un administrador de *buffer* por GPFS. Todas las entradas/salidas están protegidas por un administrador de *token*, que asegura que el sistema de archivos en múltiples nodos cumpla con la atomicidad y proporciona la consistencia de datos de los sistemas de archivos. Los demonios son procesos multi-hilo con algunos hilos dedicados a procesos específicos. Esto asegura que el servicio no se vea interrumpido porque otro hilo esté ocupado con una rutina de red. Las funciones específicas de un demonio son:

- Asignación de espacio en disco para nuevos archivos.

- Administración de directorios.

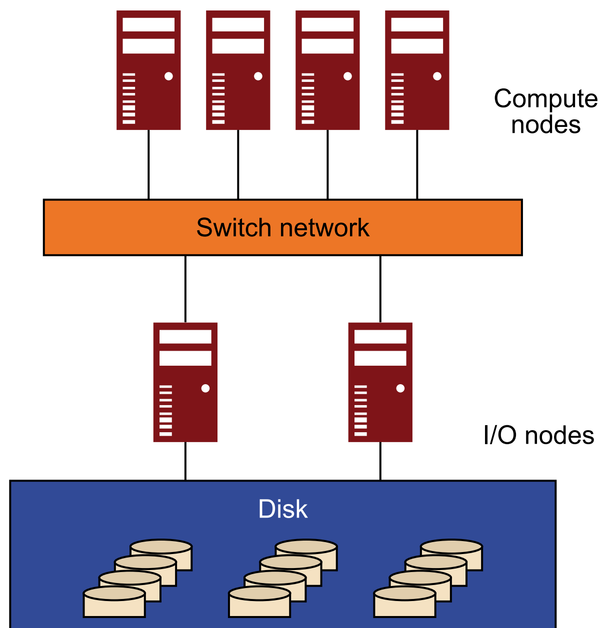
- Asignación de bloqueo para la protección de la integridad de los datos y de los metadatos.

- Los servidores de discos son iniciados con un hilo del demonio.

- La seguridad también se administra por el demonio en conjunto con el administrador de los sistemas de ficheros.

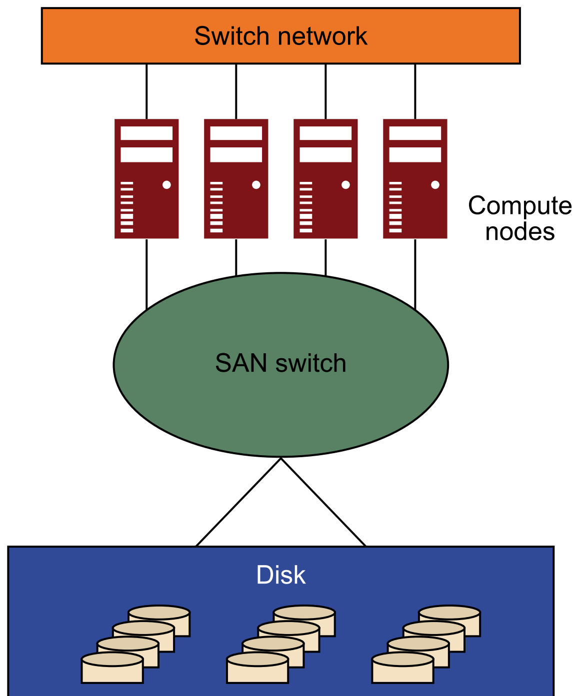
Las figuras 15 y 16 ilustran sistemas GPFS con la utilización de nodos dedicados a entrada/salida y utilizando infraestructura SAN, respectivamente.

Figura 15. Configuración de un sistema GPFS mediante nodos específicos de entrada/salida



Fuente: http://www.ncsa.illinois.edu/UserInfo/Data/filesystems/

Figura 16. Configuración de un sistema GPFS mediante SAN



Fuente: http://www.ncsa.illinois.edu/userinfo/data/filesystems/

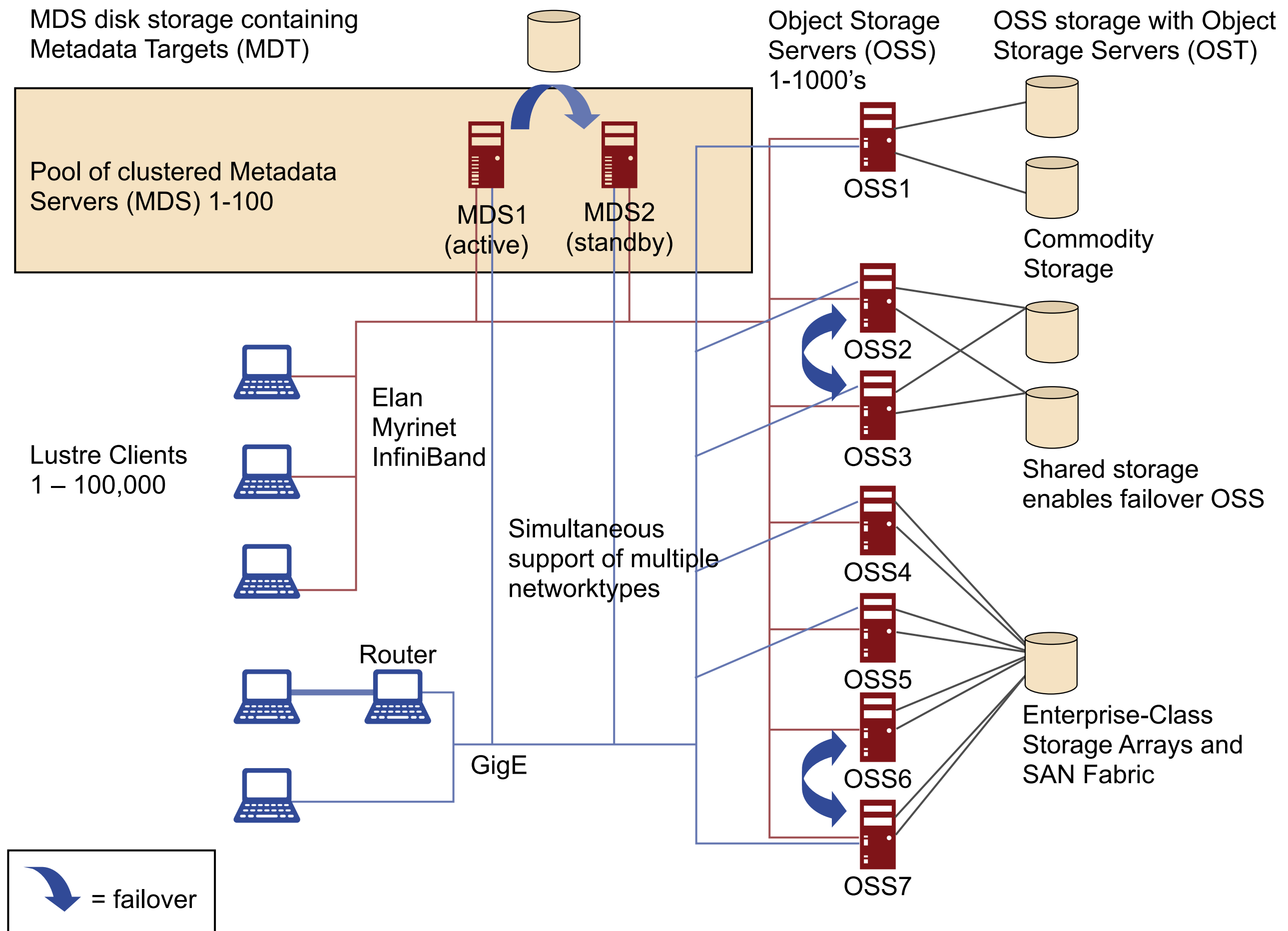
3.2.2. Linux Cluster File System (LUSTRE)

Lustre es un sistema de archivos distribuido de código abierto, normalmente utilizado en clústeres a gran escala. El nombre es una mezcla de Linux y clústeres. El proyecto intenta proporcionar un sistema de archivos por clústeres de decenas de miles de nodos con petabytes de capacidad de almacenamiento, sin comprometer la velocidad o la seguridad. Los clústeres Lustre contienen tres tipos de componentes:

- Clientes que acceden al sistema.
- *Object storage servers* (OSS), que proporcionan el servicio de entrada/salida de archivos.
- *Metadata servers* (MDS), que gestionan los nombres y los directorios del sistema de archivos.

La figura 17 muestra la estructura y organización de un clúster Lustre.

Figura 17. Estructura de un clúster Lustre a escala (fuente: www.lustre.org)



El almacenamiento de los servidores se lleva a cabo en particiones y opcionalmente organizado mediante un volumen lógico de gestión (LVM) y formateado como un sistema de ficheros. Los servidores OSS y el MDS de Lustre leen, escriben y modifican datos en el formato impuesto por este sistema. Cada OSS puede responsabilizarse de múltiples OST, uno por cada volumen, y el tráfico de entrada/salida se balancea. Dependiendo del hardware de los servidores, un OSS normalmente se encarga de 2 a 25 objetivos, cada objetivo aproximadamente de 8 terabytes de capacidad. La capacidad del sistema Lustre es la suma de las capacidades proporcionadas por los objetivos. Un OSS tiene que balancear el ancho de banda del sistema para evitar posibles cuellos de botella. Por ejemplo, 64 servidores OSS, cada uno con 2 objetivos de 8TB, proporcionan un sistema de archivos de una capacidad cercana a 1PB. Si el sistema utiliza 16 discos SATA de un terabyte, sería posible conseguir 50MB/s por cada dispositivo, proporcionando 800MB/s de ancho de banda. Si este sistema se utiliza como *back-end* de almacenamiento con un sistema de red, como por ejemplo InfiniBand, que soporta un ancho de banda similar, entonces cada OSS podría proporcionar 800MB/s al *throughput* de entrada/salida.

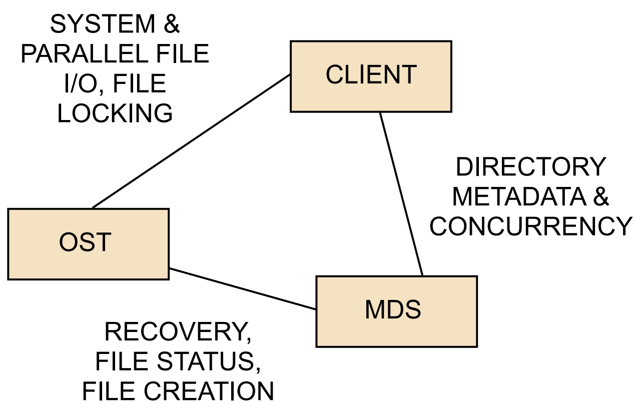
Normalmente los servidores OSS no utilizan dispositivos internos, pero utilizan una matriz de almacenamiento sobre Fiber Channel o conexiones SCSI (SAS), como en el caso de GPFS. En cualquier caso, tanto el software como el hardware RAID siguen el formato de *striping* de RAID 5 o 6. La memoria de los

OSS se utiliza como memoria caché en las lecturas de archivos y en algunos casos para escritura de datos. La utilización de la CPU es mínima cuando se utiliza RDMA.

El almacenamiento en Lustre solo se presenta en los nodos servidor, nunca a los clientes. Si se quiere utilizar un sistema más resistente a errores, se puede almacenar la información en múltiples servidores. En todo caso, el uso de SAN con *switches* caros se puede evitar puesto que las conexiones punto-a-punto entre los servidores y *arrays* de almacenamiento son la mejor elección. Para los nodos MDS se mantiene la misma consideración. Los metadatos tienen que ocupar entre el 1% -2% de la capacidad del sistema, pero el acceso a los datos para el almacenamiento MDS es ligeramente diferente al almacenamiento OSS: mientras que al primero se accede mediante metadatos, con numerosas buscas y operaciones de lectura y escritura de pequeños datos, al segundo se accede con un patrón de entrada/salida, que requiere grandes transferencias. Los sistemas Lustre son bastante sencillos de configurar.

Los clientes Lustre utilizan el sistema e interactúan con los *object storage targets* (*OST*) para la lectura/escritura de archivos y con los *metadata servers* (*MDS*). Tanto MDS, OST como OSS se pueden encontrar en el mismo nodo o en nodos diferentes. La figura 18 nos presenta el diálogo entre los diferentes elementos.

Figura 18. Diálogo entre los componentes de Lustre



Un OST proporciona almacenamiento de los objetos de los datos (*chunks*). Los objetos se presentan como *inodes* y el acceso a estos objetos es proporcionado por los OST, que realizan el servicio de entrada/salida al clúster Lustre. Los nombres son tratados por los metadatos que gestionan los *inodes*. Los *inodes* se pueden presentar como directorios, como enlaces simbólicos o dispositivos especiales, y su información y sus metadatos se almacenan en los MDS. Cuando un *inode* de Lustre representa un archivo, los metadatos hacen referencia a los objetos almacenados en los OST. En el diseño de un sistema Lustre es fundamental que los OST realicen la asignación de la información en bloques, facilitando la distribución y la escalabilidad de los metadatos. De hecho, los OST refuerzan la seguridad respecto al acceso de los objetos.

Un MDS gestiona todas las operaciones referentes a un archivo, como puede ser asignar o actualizar referencias. El MDT proporciona almacenamiento para los metadatos del sistema. Los metadatos gestionados consisten en archivos que contienen los atributos de los datos almacenados en los OST.

El servidor de gestión (MGS) define información sobre la configuración de todos los componentes presentes. Los elementos de Lustre contactan con este para proporcionar información, mientras que los clientes lo hacen para recibirla. El MGS puede proporcionar actualizaciones en la configuración de los elementos y de los clientes. El MGS necesita su propio disco de almacenamiento, pero se puede compartir un disco con un MDT. Un MGS no se considera una parte de un único sistema, sino que puede proporcionar mecanismos de configuración para otros componentes Lustre.

Los clientes son los usuarios del sistema de ficheros. Normalmente se nos presentan como nodos computacionales o de visualización. Los clientes de Lustre necesitan el software Lustre para montar el sistema de ficheros puesto que Lustre no es NFS. El cliente no es más que un software que consiste en una interfaz entre el sistema de archivos virtual de Linux y el del propio Lustre.

Los servidores y clientes se comunican los unos con los otros mediante una API de red conocida como Lustre Networking (Telnet). Telnet interacciona con una gran variedad de redes. Telnet proporciona las decisiones y los acontecimientos para cada mensaje de una conexión de red soportando *routing* entre nodos situados en diferentes redes.

Lustre se utiliza para almacenar datos, por lo que se requiere un cierto grado de redundancia para asegurar su fiabilidad. La primera idea para combatir la pérdida de información es utilizar OST redundantes en forma de espejo. Esta implementación consiste en utilizar un OST que almacenará una réplica exacta de la información, de modo que si el primer OST fallara, los objetos permanecerán en otro OST. Otra característica de esta funcionalidad es la posibilidad de balancear la carga, puesto que al disponer de múltiples copias de la misma información, la carga de lectura puede ser compartida por ambos OST.

Lustre proporciona soporte a la recuperación de un nodo cuando se presenta un fallo. Cuando Lustre se encuentra en modo de recuperación, todos sus servidores (MDS, OSS) pasan a un estado de bloqueo, es decir, la información que no ha sido guardada se mantiene en la memoria caché del cliente. Para almacenar esta información, el sistema reinicia en modo de recuperación y hace que los clientes lo escriban en el disco. En modo de recuperación, los servidores tratan de contactar con todos los clientes y contestar a sus peticiones. Si todos los clientes han sido contactados y son recuperables (no han sido reiniciados), entonces se procede a la recuperación y el sistema almacena los datos que se encuentran en la memoria caché de los clientes. Si algún cliente no es capaz de volver a conectar (debido a fallos de hardware o reinicio del cliente),

entonces el proceso de recuperación caduca, lo que provoca que los clientes sean expulsados. En este caso, si hay información en la memoria caché del cliente que no ha sido guardada, esta no se almacenará en el disco y se perderá.

4. Sistemas de gestión de colas y planificación

Tal y como se discutió en el primer módulo, los sistemas de altas prestaciones actuales son principalmente clústeres de computadores interconectados con redes de altas prestaciones. En estos sistemas, normalmente hay múltiples usuarios que quieren acceder a los recursos para ejecutar programas (paralelos) desarrollados mediante modelos de programación como los que hemos visto anteriormente (por ejemplo, *MPI*). Para organizar el acceso de los diferentes usuarios a los recursos y gestionarlos de manera eficiente (por ejemplo, para maximizar su utilización), los sistemas de altas prestaciones disponen de sistemas de gestión de recursos basados en colas.

Sistemas de altas prestaciones actuales

Estos sistemas se caracterizan porque tienen reducida latencia y gran ancho de banda de la red de interconexión.

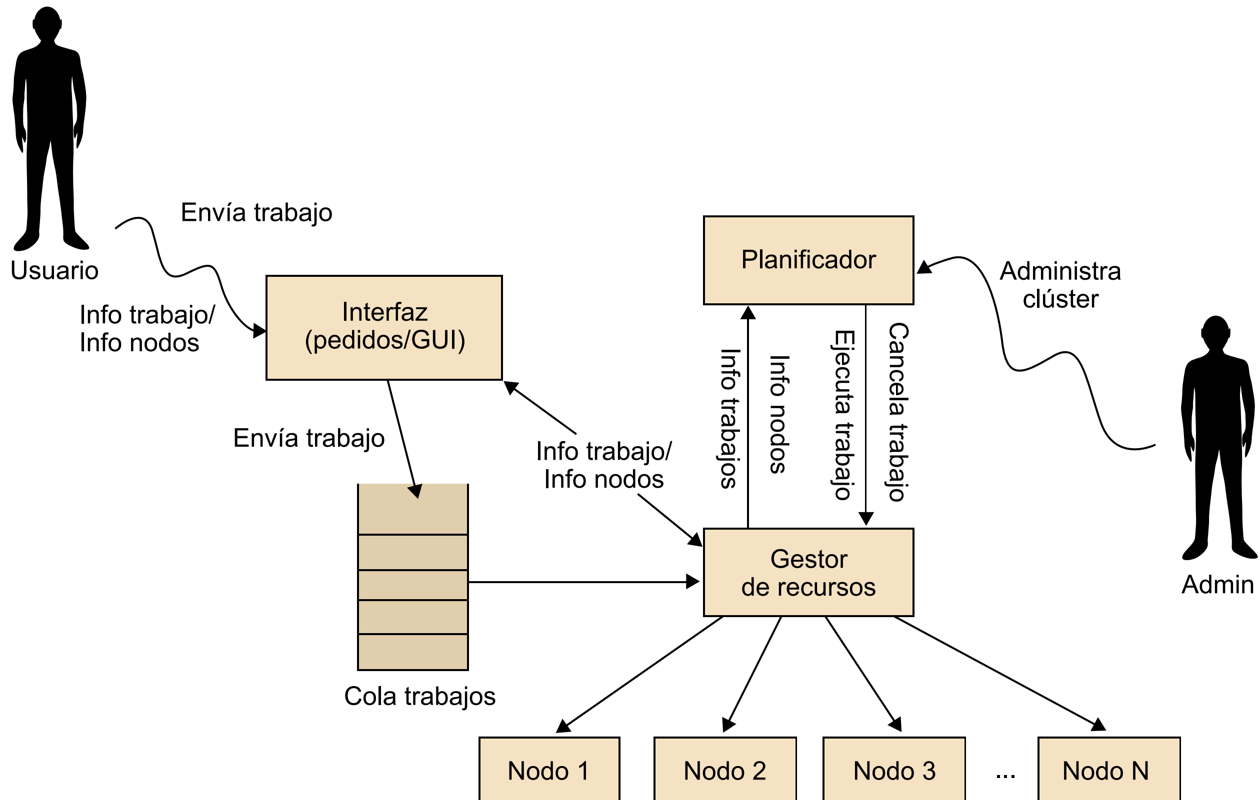
De hecho, en sistemas de altas prestaciones no tiene sentido proporcionar a los usuarios acceso a los recursos de manera interactiva, puesto que sería un caos: habría conflictos en la utilización de los recursos y poca eficiencia en la utilización y compartición de los mismos. Además, los usuarios no tienen por qué preocuparse de problemas relacionados con la gestión de los computadores, como por ejemplo el balanceo de la carga. Así pues, los sistemas de computación de altas prestaciones se basan en aplicaciones que pueden ser almacenadas en una cola y ejecutadas posteriormente en segundo plano. Estos sistemas se denominan tradicionalmente sistemas por lote (*batch*).

Ejemplos de conflictos

Algunos usuarios podrían hacer un uso desmesurado de los recursos, o incluso procesos erróneos podrían malgastar los recursos sin que fuera posible controlarlo.

La figura 19 muestra un esquema de un clúster de altas prestaciones con los componentes esenciales para su gestión. También hay que tener en cuenta que en los sistemas de altas prestaciones normalmente encontramos el gran grueso de nodos que se encargan de la computación, pero normalmente no se puede acceder directamente a los mismos por medidas de seguridad y gestión, y hay un conjunto de nodos (que pueden ser de las mismas o diferentes características que el resto de los nodos) que se utilizan como puerta de entrada (*login nodes*). Estos nodos de acceso normalmente posibilitan el desarrollo de las aplicaciones, disponen de sistemas completos con herramientas de compilación, depuración, etc. y permiten a los usuarios hacer el envío de sus trabajos a los sistemas de colas para su ejecución.

Figura 19. Esquema de un sistema clúster con los componentes de gestión de recursos



4.1. Sistemas de gestión de colas

Los sistemas de gestión de colas son sistemas de administración, planificación y ejecución de trabajos que se envían a un clúster. A continuación se exponen brevemente los sistemas de gestión de colas *PBS*, que en la actualidad son los más habituales en la computación de altas prestaciones.

4.1.1. PBS (*portable batch system*)

El *PBS*[6] es un sistema de colas diseñado originalmente por la NASA y que ha llegado a ser uno de los sistemas de colas más populares en la computación de altas prestaciones. De hecho, un gran porcentaje de sistemas de altas prestaciones actuales utilizan *PBS* como sistema de colas.

(6)Del inglés *portable batch system*.

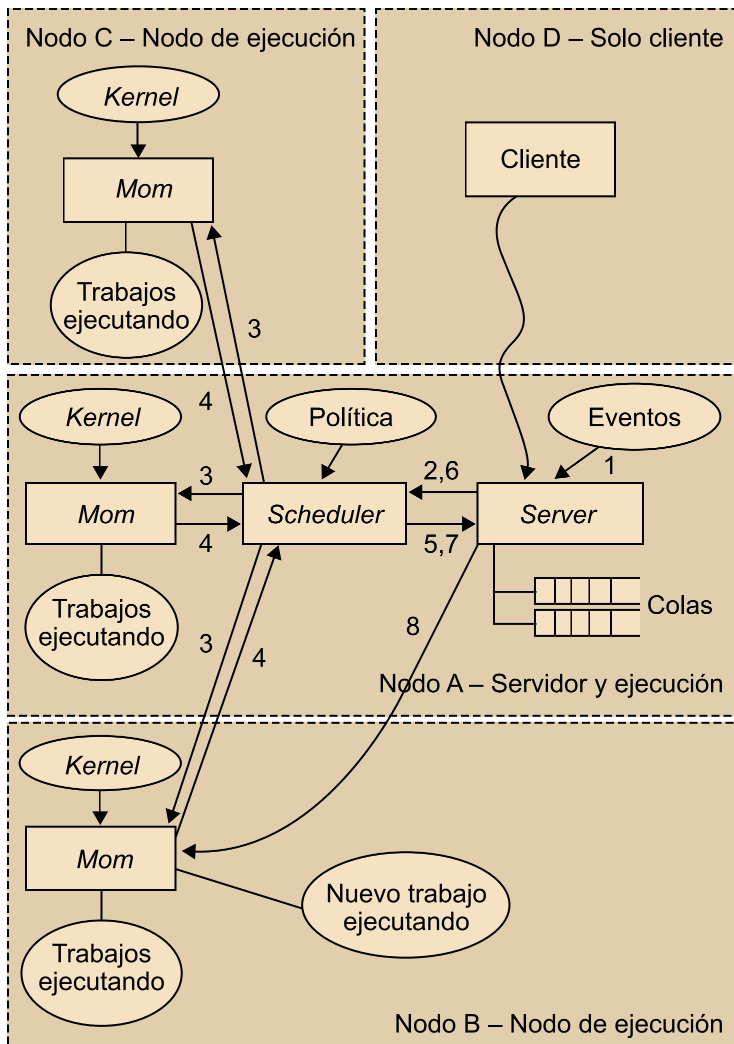
PBS proporciona una serie de herramientas para la gestión de trabajos *batch*, utilizando una unidad de programación de tareas. Además, permite la planificación de trabajos en diferentes computadores y también definir e implementar políticas sobre la utilización de estos recursos.

La figura 20 muestra el esquema del sistema de colas, en el que podemos diferenciar los componentes que lo forman y las interacciones entre los mismos. Podemos distinguir claramente tres componentes básicos.

- *server*: es un demonio encargado de recibir los trabajos que hay que ejecutar y esperar los comandos del usuario.

- *mom* (o *job executor*): es un demonio que se ejecuta en cada uno de los nodos de cómputo y que envía y recibe los trabajos que hay que ejecutar.
- *scheduler*: es un planificador que decide qué trabajos se tienen que ejecutar en función de la política que esté establecida.

Figura 20. Esquema del sistema de colas *PBS*



Estos componentes interactúan mediante los pasos que se describen a continuación, de una manera cíclica:

- Un evento le dice al *server* que empiece un ciclo de planificación.
- El *server* le envía una petición de planificación al *scheduler*.
- El *scheduler* solicita información sobre los recursos a los *mom*.
- Los *mom* retornan la información solicitada al *scheduler*.
- El *scheduler* solicita información sobre el trabajo al *server*.
- El *server* retorna la información sobre el estado del trabajo al *scheduler* y toma la decisión y ejecuta el trabajo o no en función de la política establecida.
- El *scheduler* envía una petición de ejecución al *server*.
- El *server* envía el trabajo al *mom* para que lo ejecute.

Los usuarios interactúan con el sistema de colas mediante una serie de instrucciones que se pueden llamar desde el sistema operativo.

Para ejecutar un programa, el usuario tiene que definir un trabajo mediante un fichero *script*. Además de indicar cuál es el programa que hay que ejecutar y los argumentos, en este fichero el usuario permite lo siguiente.

- Especificar si es un trabajo *batch* o interactivo. El hecho de que sea interactivo no quiere decir que se ejecute de manera inmediata, sino que cuando se ejecuta le permite al usuario interactuar con el programa, a diferencia de los trabajos *batch*, que se ejecutan en segundo plano sin que el usuario pueda interactuar directamente.

- Definir una lista con los recursos necesarios.

- Definir la prioridad del trabajo para su ejecución.

- Definir el tiempo de ejecución, que es una información importante para planificar los trabajos de modo eficiente.

- Especificar si se quiere enviar un correo electrónico al usuario cuando la ejecución empieza, acaba o es abortada.

- Definir dependencias.

- Sincronizar el trabajo con otros trabajos.

- Especificar si se quiere hacer *checkpointing* (en el caso de que el sistema operativo ofrezca esta posibilidad). El *checkpointing* resulta especialmente importante en ejecuciones muy largas en las que se desea monitoritzar la aplicación durante el tiempo de ejecución, o bien por cuestiones de seguridad hacia posibles fallos.

Además de garantizar que el trabajo se podrá ejecutar, la lista de recursos que es posible especificar en un fichero de definición de un trabajo permite mejorar la utilización del sistema, puesto que el planificador puede tomar decisiones más cuidadosas. A continuación, se describe una lista con los recursos más comunes (no completa).

Decisiones cuidadosas

Un ejemplo sería no dejar núcleos sin trabajo porque no hay bastante memoria en un nodo para asignarle otro trabajo.

- *cput*: máximo tiempo de *CPU* usado por todos los procesos del trabajo.
- *pcput*: máximo tiempo de *CPU* usado por cada uno de los procesos del trabajo.
- *mem*: cantidad máxima de memoria física utilizada por un trabajo.
- *pmem*: cantidad máxima de memoria física utilizada por cada uno de los procesos de un trabajo.
- *vmem*: cantidad máxima de memoria virtual utilizada por un trabajo.

- *pvmem*: cantidad máxima de memoria virtual utilizada por cada uno de los procesos de un trabajo.
- *walltime*: tiempo de ejecución (de reloj, no de *CPU*).
- *file*: tamaño máximo de cualquier fichero que puede crear el trabajo.
- *host*: nombre de los nodos computacionales en los que ejecutar el trabajo.
- *nodes*: número y/o tipo de los nodos que hay que reservar para el uso exclusivo del trabajo.

PBS facilita una interfaz para el intérprete de comandos y una interfaz gráfica. Aun así, lo más habitual en la computación de altas prestaciones es utilizar la interfaz a través del intérprete de comandos para enviar, monitorizar, modificar y eliminar trabajos. Hay varios tipos de comandos en función del tipo de usuario que los usa.

- Comandos de usuario: `qsub`, `qstat`, `qdel`, `qselect`, `qrerun`, `qorder`, `qmove`, `qhold`, `qalter`, `qmsg`, `qrls`.
- Comandos de operación: `qenable`, `qdisable`, `qrun`, `qstart`, `qstop`, `qterm`.
- Comandos de administración: `qmgr`, `pbsnodes`.

A continuación, veremos la sintaxis de algunos de los comandos más utilizados desde el punto de vista del usuario en *PBS*:

```
qsub [-a date_time] [-A account_string] [-c interval]
   [-C directive_prefix] [-e path] [-h] [-I] [-j join]
   [-k keep] [-l resource_list] [-m mail_options]
   [-M user_list] [-N name] [-o path] [-p priority]
   [-q destination] [-r c] [-S path_list] [-u user_list][-v variable_list] [-V]
   [-W additional_attributes] [-z] [script]
```

El comando `qsub` envía un nuevo trabajo al servidor de *PBS* para su ejecución. Por defecto, se considera que el trabajo se encolará y se ejecutará en segundo plano, pero igualmente se permite ejecutar el trabajo (también cuando el planificador lo considere oportuno) de manera interactiva, si se especifica la opción `-q`. De manera típica, el `script` que se envía a través del comando es un *shell script* del intérprete de comandos, como por ejemplo `sh` o `csh`.

Las opciones del comando `qsub` que se indican anteriormente permiten especificar atributos que afectarán al comportamiento del trabajo. Además, el comando `qsub` le pasará al trabajo una lista de variables de entorno que estarán disponibles durante la ejecución del trabajo. El valor de las variables `HOME`, `LANG`, `LOGNAME`, `PATH`, `MAIL`, `SHELL` y `TZ` se toma de las variables de entorno del comando `qsub`. Estos valores se asignarán a variables que empiezan por `PBS_O_`. Por ejemplo, el trabajo tendrá acceso a una variable de entorno de-

nominada PBS_O_HOME que tendrá el valor de la variable HOME en el entorno de qsub. Aparte de las variables de entorno anteriores, también encontramos las siguientes disponibles para el trabajo.

- PBS_O_HOST: es el nombre del servidor en el que el comando qsub se está ejecutando.
- PBS_O_QUEUE: es el nombre de la cola inicial a la que se envió el trabajo.
- PBS_O_WORKDIR: es la ruta (absoluta) del directorio de trabajo que se ha indicado en el comando qsub.
- PBS_ENVIRONMENT: indica si el trabajo es en segundo plano (PBS_BATCH) o interactivo (PBS_INTERACTIVE).
- PBS_JOBID: es el identificador de trabajo que le asigna el sistema de colas.
- PBS_JOBNAME: se trata del nombre del trabajo proporcionado por el usuario.
- PBS_NODEFILE: es el nombre del fichero que contiene la lista de nodos que se le asignará al trabajo.
- PBS_QUEUE: se trata del nombre de la cola desde la que el trabajo se ha ejecutado.

```
qstat [-f][-W site_specific]
   [job_identifier... | destination...]

qstat [-a|-i|-r] [-n] [-s] [-G|-M] [-R] [-u user_list][job_identifier... | destination...]

qstat -Q [-f][-W site_specific] [destination...]

qstat -q [-G|-M] [destination...]

qstat -B [-f][-W site_specific] [server_name...]
```

El comando qstat indica el estado de un trabajo, de las colas y del servidor de colas. El estado se proporciona mediante la salida estándar. Cuando se solicita el estado de un trabajo mediante una de las dos opciones mostradas, qstat retorna la información de cada uno de los trabajos indicados a través de su identificador o todos los trabajos de un destino dado. Hay que tener presente que los trabajos de los que el usuario no tiene privilegios no se mostrarán. Cuando se utiliza una de las tres últimas opciones mostradas, qstat retornará la información de cada destino.

```
qdel [-W delay] job_identifier ...
```

El comando qdel elimina los trabajos especificados en el comando en el orden en el que se proporcionan. Un trabajo solo puede ser eliminado o bien por su propietario o bien por el operador o el administrador del sistema de colas. Para eliminar un trabajo, el servidor enviará una señal SIGTERM seguida por una señal SIGKILL.

```
qalter [-a date_time] [-A account_string] [-c interval]
   [-e path] [-h hold_list] [-j join] [-k keep]
   [-l resource_list] [-m mail_options] [-M user_list]
   [-N name] [-o path] [-p priority] [-r c] [-S path]
   [-u user_list] [-W additional_attributes]
   job_identifier...
```

El comando `qalter` modifica los atributos de uno o varios trabajos a partir de su identificador. Solo es posible modificar las opciones que se muestran en la descripción del comando.

El envío de trabajos se lleva a cabo mediante *scripts* que permiten comunicarse con el programa de envío de trabajos *MPI* para especificarle la cantidad y qué nodos se tienen que utilizar de los disponibles para *PBS*, según los requerimientos del usuario. El server de *PBS* no ejecutará más trabajos en los nodos ocupados hasta que se haya acabado el trabajo actual. A continuación, se muestra un ejemplo de *script* de *PBS* para la ejecución de una aplicación *MPI* mediante `mpiexec` (equivalente a `mpirun`, que vimos anteriormente).

```
#!/bin/sh
#! Ejemplo de fichero de definición de trabajo para enviar mediante qsub
#! Las líneas que empiezan por #PBS son options de la orden qsub

#! Número Número de procesos (8 en este caso, 4 por nodo)
#PBS -l nodos=4:ppn=2

#! Nombre de los ficheros para la salida estándar y error
#! Si no se especifican, por defecto son <job-name>.o <job_number> y <job-name>.e<job_number>
#PBS -e test.err
#PBS -o test.log

#! Dirección de correo electrónico del usuario para cuando el trabajo acabe o se aborte
#PBS -m ae

#! Directorio de trabajo
echo Working directory is $PBS_O_WORKDIR
#!cd <working directory>
echo Running on host 'hostname'
echo Time is 'date'
echo Directory is 'pwd'
echo This jobs runs on the following processors:
echo 'cat $PBS_NODEFILE'

MPI executable - it's possible to redirect stdin/stdout of all processes
#! using "<" and ">" - including the double quotes
/usr/local/bin/mpiexec -bg a.out
```

También hay otros gestores de colas, como por ejemplo *Loadleveler* o SLURM. *Loadleveler* es un gestor de colas diseñado por IBM y que se caracteriza por su facilidad para procesar trabajos en entornos clúster. Permite la ejecución de trabajos paralelos (por ejemplo, *MPI*) y es fácilmente escalable en miles de procesadores. *Loadleveler* fue uno de los primeros sistemas en incorporar el algoritmo de planificación *backfilling*. SLURM es un gestor de colas de código abierto diseñado para clústeres Linux de diferentes tamaños.

4.2. Planificación

El problema de la planificación de tareas o trabajos en un sistema de computación de altas prestaciones se puede definir como, dada una lista de trabajos que están esperando en la cola, sus características (por ejemplo, el número de procesadores que necesita, los ficheros ejecutables, los ficheros de entrada, etc.) y el estado del sistema, decidir qué trabajos se tienen que ejecutar y en qué procesadores. Este problema se puede dividir en dos etapas que siguen, cada una, una política concreta: la política de planificación de trabajos y la política de selección de recursos.

Trabajo

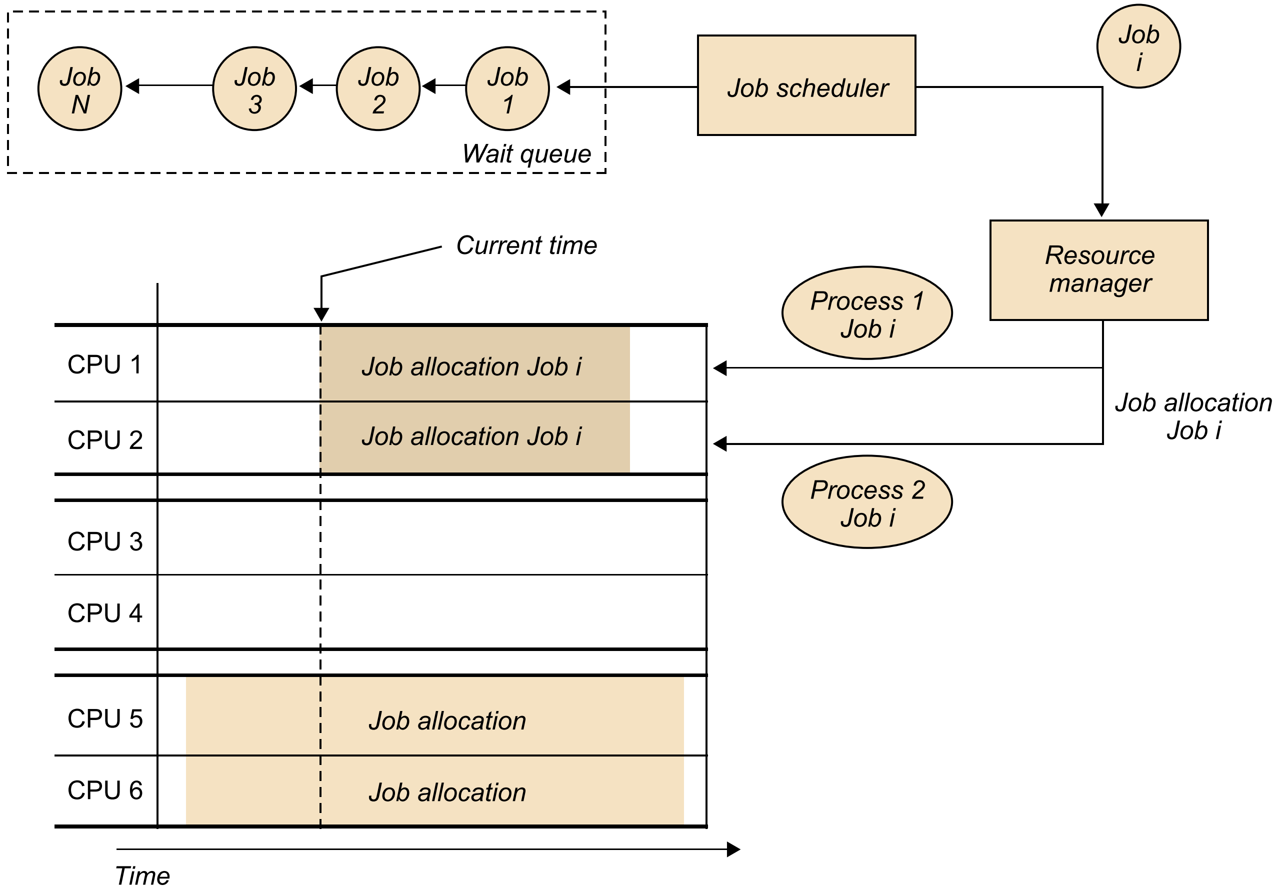
Durante todo el subapartado, utilizaremos el término *trabajo* para referirnos a la ejecución de una aplicación con unos parámetros concretos.

Como muestra la figura 21, en la primera etapa de la planificación el planificador (*scheduler*) tiene que decidir cuál de los trabajos que esperan en la cola (*job1*, *job2*, ..., *jobN*) debe empezar, teniendo en cuenta los recursos que hay disponibles (*CPU1*, *CPU2*, *CPU3*, *CPU4*). El algoritmo que utiliza el planificador para hacer esta selección se denomina política de planificación de trabajos[7]. Una vez se ha seleccionado el trabajo más apropiado para ser ejecutado, el planificador le pide al gestor de recursos (*resource manager*) que asigne el trabajo seleccionado (*jobi*) a los procesadores más adecuados en función de los requerimientos del trabajo (en la figura 21, el trabajo *jobi* requiere dos procesadores). Entonces, el gestor de recursos seleccionará los procesadores más apropiados (*CPU1* y *CPU2* en el ejemplo) y asignará procesos del trabajo a cada uno de los mismos. El gestor de recursos implementa una política de selección de recursos[8] concreta.

(7)En inglés, *job scheduling policy*.

(8)En inglés, *resource selection policy*.

Figura 21. El problema de la planificación mediante un ejemplo



Se han propuesto muchas técnicas y políticas de planificación y de selección de recursos durante las últimas décadas para optimizar la utilización de los recursos y reducir los tiempos de respuesta de los trabajos. Tal y como veremos a continuación, las políticas de planificación de trabajos que más se han utilizado en entornos de altas prestaciones son las que están basadas en *backfilling*. Se ha demostrado que estas políticas proporcionan un buen balance entre el rendimiento del sistema (número de procesadores utilizados) y el tiempo de respuesta de los trabajos. Aun así, el principal problema del algoritmo que utilizan estas políticas consiste en que el usuario tiene que proporcionar una estimación del tiempo de ejecución de los trabajos que envía y, normalmente, estas estimaciones no son nada cuidadosas. Además, en muchas situaciones el usuario no tiene suficientes conocimientos o información para proporcionar estos parámetros.

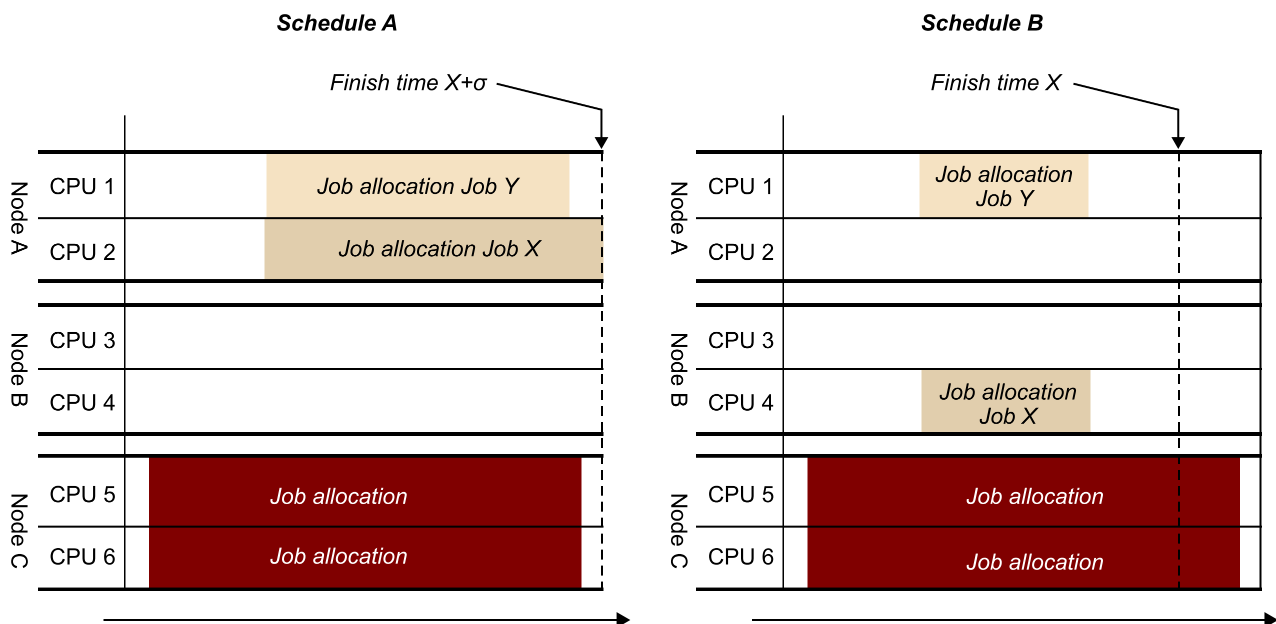
Tiempo de respuesta del trabajo

Es el tiempo desde que se envía un trabajo hasta que acaba su ejecución incluyendo el tiempo en la cola.

Por ejemplo, un usuario no experto en arquitectura de computadores no tiene por qué saber que una aplicación tarda una cierta duración en un tipo de arquitectura o un 25% más en otro tipo concreto que inicialmente parece similar (por ejemplo, debido a la cantidad de memoria caché). A pesar de todo, el tiempo de ejecución de un trabajo puede depender de muchos factores como, por ejemplo, el número de procesadores empleados, el estado del sistema, los datos de entrada del programa, etc. Es posible que también suceda que usuarios con experiencia no dispongan de suficiente información para estimar el tiempo de ejecución. Las políticas de planificación de trabajos se centran en la selección de los trabajos que se tienen que ejecutar, pero muchas veces no tienen en cuenta la ubicación que deben tener. Esta decisión se toma muchas veces de manera unilateral mediante políticas de selección de recursos.

Normalmente, lo que hace el sistema de recursos es ubicar los trabajos de dos maneras distintas: o bien un conjunto de filtros predefinidos se aplican a las características de los trabajos para encontrar los procesadores adecuados, o bien los procesos de los trabajos se ubican en nodos completos cuando el usuario especifica que los trabajos tienen que ejecutarse en nodos no compartidos. La primera forma no tiene en cuenta el estado del sistema y, en consecuencia, un trabajo intensivo de memoria podría ser ubicado en un nodo en el que el ancho de banda en memoria está prácticamente saturado, lo que causaría una importante degradación de todos los trabajos ubicados en el nodo. La segunda forma normalmente implica que la utilización de los recursos se reduce de manera sustancial, ya que normalmente los trabajos no utilizan todos los procesadores y recursos del nodo donde están ubicados. Por lo tanto, en ciertas arquitecturas, cuando los trabajos comparten recursos (por ejemplo, la memoria) pueden producir sobrecarga. Esto tiene un impacto negativo en el rendimiento de los trabajos que utilizan los recursos, y hay también un impacto negativo colateral en el rendimiento del sistema. Este problema se ilustra en la figura 22, en la que se muestran diferentes planificaciones de los trabajos cuando los trabajos *job Y* y *job X* se envían al sistema. Teniendo en cuenta las políticas de planificación actuales, la planificación más probable sería el caso A. En esta, los trabajos *job Y* y *job X* se ubican de manera consecutiva en el mismo nodo (nodo A). Si los dos trabajos son intensivos en memoria, habría una reducción en el rendimiento de su ejecución debido a la contención de recursos. En cambio, si el planificador tiene en cuenta el estado del sistema y los requerimientos de los trabajos, la planificación más probable sería la del caso B. En este último caso, tanto el trabajo *X* como *Y* no tendrían penalización en el rendimiento de su ejecución por contención de recursos.

Figura 22. El problema de la utilización de recursos

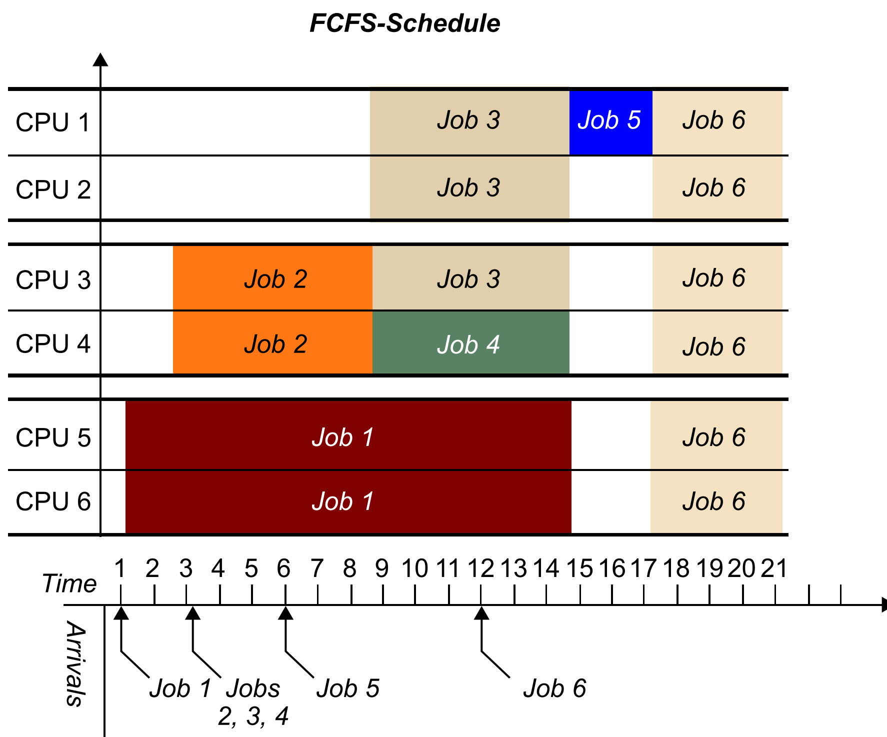


4.2.1. Políticas de planificación basadas en *backfilling*

Desde principios de los años noventa, se ha investigado de manera muy activa en el ámbito de la planificación de trabajos en computadores de altas prestaciones, y las políticas basadas en *backfilling* son las que se han utilizado más extensamente en centros de altas prestaciones. *Backfilling* es una optimización de la política de planificación *FCFS*[9]. En *FCFS*, el planificador encola todos los trabajos enviados en orden de llegada y cada uno de estos trabajos tiene asociado un número de procesadores requeridos. Cuando un trabajo finaliza su ejecución, si hay suficientes recursos para empezar la ejecución del siguiente trabajo de la cola, el planificador lo toma y empieza su ejecución. En caso contrario, el planificador tiene que esperar hasta que haya bastantes recursos disponibles y ningún otro trabajo de la cola puede ser ejecutado. La figura 23 muestra una posible planificación de la política *FCFS*. El principal problema de utilizar esta política es que el sistema sufre fragmentación y, además, la utilización de los recursos puede ser muy baja.

(9)Del inglés *first-come-first-serve*.

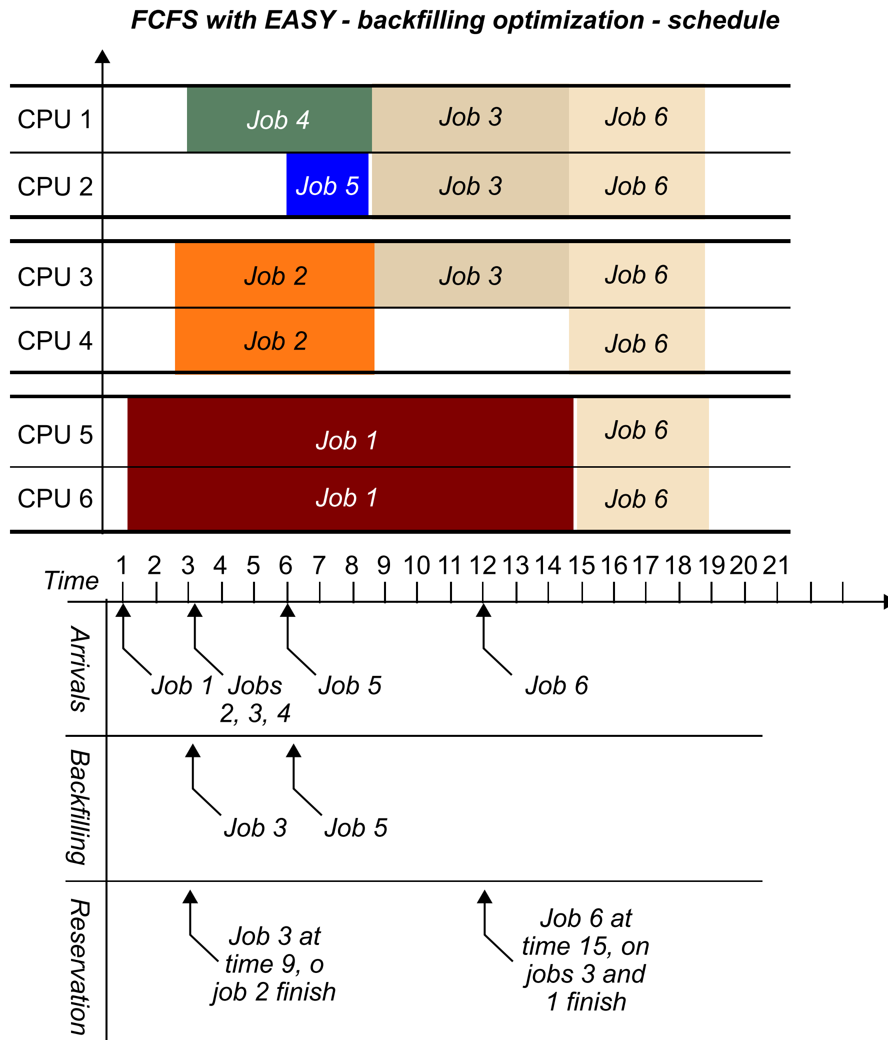
Figura 23. Ejemplo de planificación basada en la política *FCFS*



Con la política de planificación *backfilling*, se permite ejecutar trabajos que se han "encolado" posteriormente a otros que ya están en la cola sin que se atrase el tiempo estimado de ejecución de estos trabajos que ya estaban "en espera". Notad que para aplicar esta optimización hay que saber el tiempo de ejecución aproximado de los trabajos cuando se envía (por lo tanto, antes de su ejecución), puesto que de no ser así, no hay suficiente información para asegurar que el tiempo de ejecución del trabajo del principio de la cola no se atrase. La figura 24 presenta una posible planificación del ejemplo anterior con optimización mediante una política *backfilling*. En este ejemplo, a los trabajos *job 4* y *job 5* se les ha aplicado *backfilling* porque los trabajos *job 3* y *job 6* no

pueden empezar, puesto que no hay suficientes procesadores disponibles para los mismos. Gracias a esta optimización, la utilización del sistema se puede mejorar de manera muy significativa. Muchos de los procesadores que quedarían parados sin trabajo, con una política *FCFS* pueden adelantarse a partir de *backfilling*.

Figura 24. Ejemplo de planificación *FCFS* optimizada con *EASY-backfilling*

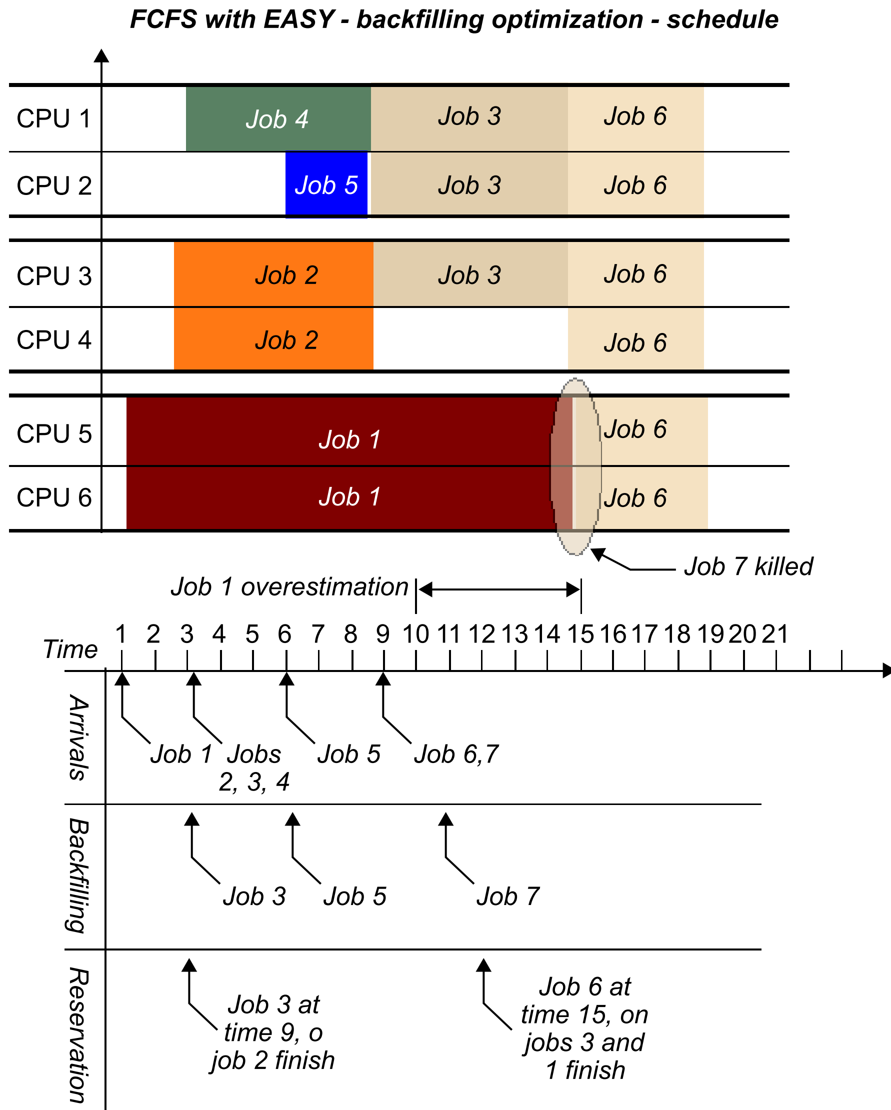


El principal inconveniente de este algoritmo es que el usuario tiene que proporcionar el tiempo de ejecución estimado por el trabajo cuando este se envía al sistema de colas. En situaciones en las que el tiempo de ejecución proporcionado por el usuario es inferior al tiempo real de ejecución, el planificador mata el trabajo cuando detecta que ha excedido el tiempo de ejecución solicitado. La otra situación que sucede con frecuencia en este escenario consiste en que el tiempo de ejecución estimado es sustancialmente superior al real.

En el ejemplo de la figura 25, se muestran las dos situaciones descritas anteriormente. En esta planificación, el trabajo *job* 1 ha sido sobrestimado. Una vez el planificador ha detectado esta situación, utiliza la estimación de tiempo de ejecución proporcionada por el usuario y aplica *backfilling* al trabajo *job* 7.

Aun así mata el trabajo *job* 7 más adelante, puesto que excede el tiempo de ejecución especificado. Observad que en el caso de que el trabajo *job* 7 no fuera matado, el tiempo de inicio del trabajo *job 6* podría atrasarse.

Figura 25. Ejemplo de sobreestimación y subestimación de recursos en planificación *FCFS* optimizada con *EASY-backfilling*



La política *backfilling* que hemos presentado anteriormente es la política basada en *backfilling* más sencilla, que fue propuesta por Lifka y otros y se denomina *EASY-backfilling*. Se han propuesto muchas variantes de esta política, de las cuales podemos destacar las que tienen en cuenta, por ejemplo, el orden en el que se puede aplicar *backfilling* a los trabajos. Un ejemplo de esto es la variante *Shortest-Job-Backfilled-First*.

Lectura complementaria

J. Skovira; W. Chan; H. Zhou; D. A. Lifka (1996). *"The easy-loadleveler api project. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing". Lecture Notes In Computer Science* (vol. 1162, págs. 41-47).

Actividad

Buscad información sobre diferentes políticas y técnicas de planificación de trabajos, tomando como referencia el trabajo del doctor Feitelson.

Bibliografía

Buyya, R. (1999). *High Performance Cluster Computing: Architectures and Systems* (vol. 1). Prentice Hall.

Grama, A.; Karypis, G.; Kumar, V.; Gupta, A. (2003). *Introduction to Parallel Computing* (2.ª ed.). Addison-Wesley.

Hwang, K.; Fox, G. C.; Dongarra, J. J. (2012). *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann.

Jain, R. (1991). *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. Wiley-Interscience.

Lin, C.; Snyder, L. (2008). *Principles of Parallel Programming*. Addison Wesley.

Pacheco, P. (2011). *An Introduction to Parallel Programming* (1.ª ed.). Morgan Kaufmann.

Skovira, J.; Chan, W.; Zhou, H.; Lifka, D. A. (1996). "The easy-loadleveler api project. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing". *Lecture Notes In Computer Science* (vol. 1162, págs. 41-47).