Organització i gestió de sistemes d'altes prestacions

Ivan Rodero Castro
Francesc Guim Bernat

PID_00218846

UOC
Universitat Oberta de Catalunya

Índex

Introducció

En aquest primer mòdul didàctic estudiarem l'organització i gestió dels sistemes d'altes prestacions. També es presentaran les característiques dels components principals dels sistemes d'altes prestacions com són la xarxa d'interconnexió i sistema d'arxius d'altes prestacions.

Finalment s'introduirà l'entorn d'execució d'aplicacions paral·leles per a computació d'altes prestacions. Estudiarem els components de programari utilitzats en els sistemes paral·lels orientats a les altes prestacions i també polítiques de planificació que permeten optimitzar l'execució de les aplicacions paral·leles i la utilització dels sistemes d'altes prestacions.

Objectius

Els materials didàctics d'aquest mòdul contenen les eines necessàries per a assolir els objectius següents:

1. Entendre el funcionament i organització dels sistemes d'altes prestacions.

2. Conèixer els fonaments dels components bàsics dels sistemes d'altes prestacions.

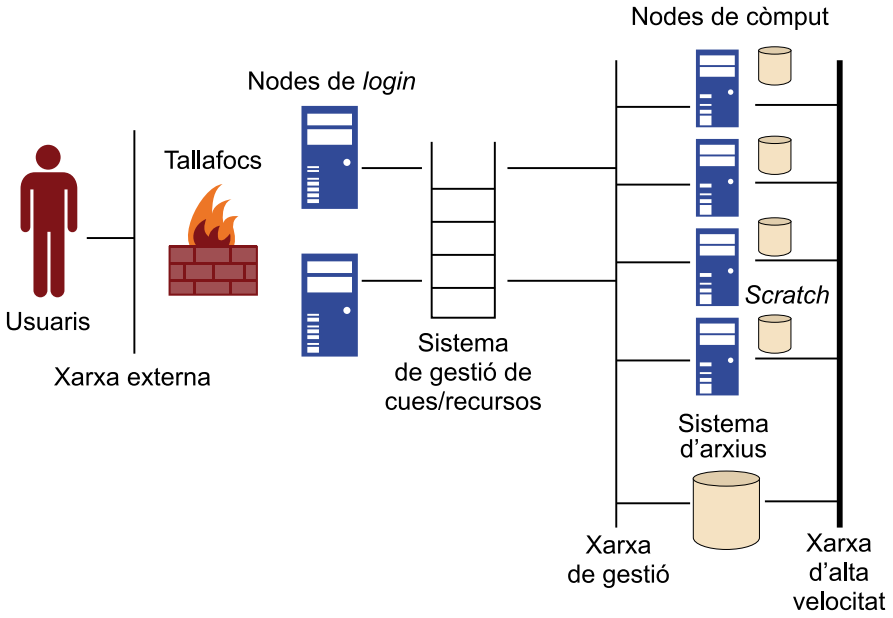
3. Conèixer el funcionament i els components que formen els sistemes de gestió d'aplicacions en sistemes paral·lels per a computació d'altes prestacions.

4. Aprendre els conceptes fonamentals de les polítiques de planificació de treballs en entorns d'altes prestacions.

1. Organització dels sistemes d'altes prestacions

Tal com vam veure en el primer mòdul didàctic d'aquesta assignatura, actualment els sistemes d'altes prestacions o supercomputadors són sistemes de tipus clúster. Anteriorment els sistemes de memòria compartida i els multiprocessadors havien estat populars però la necessitat de més nivell de paral·lelisme i l'aparició de xarxes d'interconnexió d'altes prestacions han fet que els sistemes d'altes prestacions actuals siguin un conjunt de computadors independents interconnectats, que treballen conjuntament per tal de resoldre un problema. Així doncs, els computadors i la xarxa de comunicació són elements essencials en un sistema d'altes prestacions però també hi ha altres elements clau tal com mostra la figura 1.

Figura 1. Elements bàsics d'un sistema d'altes prestacions típic



Els computadors dels sistemes d'altes prestacions s'organitzen principalment en computadors (o nodes) de dos tipus diferents:

- Nodes de *login*: són els nodes accessibles pels usuaris, habitualment mitjançant connexions segures de tipus `ssh`. Els supercomputadors normalment disposen de diferents nodes de *login*. Aquests tipus de nodes faciliten funcions bàsiques com ara crear i editar fitxers, compilar codi font, utilitzar eines per a estudiar el rendiment d'aplicacions, fer proves de funcionalitat, enviar i gestionar treballs al sistema de cues, etc. En alguns casos el nodes de *login* són del mateix tipus que els nodes de còmput però poden ser de diferents tipus i fins i tot tenir una arquitectura diferent. En el darrer cas, l'usuari ha d'utilitzar opcions específiques de compilació perquè el programa pugui executar-se correctament en els nodes de còmput. Com

a part de la infraestructura de seguretat del sistema també s'acostumen a utilitzar tallafocs per a aïllar el sistema de la xarxa externa.

- Nodes de còmput: són els que executen les aplicacions paral·leles utilitzant una xarxa d'alta velocitat. Tot i que poden ser heterogenis, normalment són homogenis o bé tenen diferents particions que són homogènies. Per exemple, pot haver-hi un conjunt de nodes amb CPU, d'altres que incloguin GPU, i d'altres que s'utilitzin per a la visualització i tinguin característiques específiques com ara més capacitat de memòria.

Un dels elements essencials dels sistemes d'altes prestacions i que els diferencien d'altes sistemes com ara centres de dades és la xarxa d'interconnexió. Normalment s'hi pot trobar una xarxa d'alta velocitat, és a dir, de gran amplada de banda però amb latències molt reduïdes i una altra de gestió que no intervé en l'execució de les aplicacions paral·leles (en el pas de missatges). Alguns exemples de xarxes d'altes prestacions són Infiniband, Cray Gemini o Myrinet. Les característiques de diferents tipus de xarxes les estudiarem més endavant en aquest mòdul didàctic.

Els sistemes d'altes prestacions també acostumen a utilitzar un sistema d'arxius paral·lel compartit per tots els nodes del sistema. Tal com veurem més endavant, aquest tipus de sistema permet millorar el rendiment i també altres qüestions com ara l'escalabilitat amb un nombre molt elevat de nodes o utilitzar grans volums de dades. De manera complementaria, els nodes poden tenir un espai d'emmagatzematge temporal (conegut com a *scratch*), per exemple, en forma de disc o SSD (*solid-state drive*) per a millorar la localitat de les dades quan no cal compartir-les. D'aquesta manera es pot reduir la utilització de la xarxa d'interconnexió i del sistema d'arxius paral·lel però cal tenir en compte que al final de l'execució al moure les dades a un suport d'emmagatzematge permanent. Finalment, també es poden trobar dispositius de còpia de seguretat o d'emmagatzemament massiu com per exemple els basats en cintes que ofereixen gran capacitat però amb molta més latència.

Per a poder utilitzar els recursos eficientment i compartir-los entre múltiples usuaris, els sistemes d'altes prestacions fan servir sistemes de cues com a interfície entre l'usuari i els nodes de còmput. A més de mantenir els treballs dels usuaris del sistema, també els gestionen mitjançant polítiques de planificació tal com veurem més endavant en aquest mòdul didàctic.

Respecte al programari de sistema cal destacar que actualment els sistemes d'altes prestacions acostumen a utilitzar sistemes basats en Unix, especialment Linux. A més, també inclou *middleware*, el qual té els objectius següents:

- Proporcionar transparència a l'usuari de manera que no s'hagi de preocupar dels detalls de baix nivell.

- Escalabilitat del sistema.

- Disponibilitat del sistema per a suportar les aplicacions dels usuaris.

- El concepte d'SSI (*single system image*) permet a l'usuari veure el clúster de manera unificada com si fos un únic recurs o sistema de còmput.

Des del punt de vista de l'entorn d'execució i desenvolupament, el programari, com ara compiladors i paquets matemàtics, s'acostuma a gestionar per mitjà de mòduls. Això permet modificar l'entorn de manera dinàmica (per exemple, utilitzar una certa versió d'un compilador). Algunes ordres útils quan s'utilitzen mòduls són consultar els mòduls disponibles (`module avail`), carregar un mòdul (`module load<aplicació>`) o descarregar-ne (`module unload<aplicació>`).

En els pròxims capítols estudiarem les característiques bàsiques dels elements que formen els sistemes d'altes prestacions i els tipus més comuns que es troben en l'actualitat.

2. Xarxes d'interconnexió

Tal i com s'ha comentat, la xarxa d'interconnexió juga un paper decisiu en el rendiment de tant sistemes de memòria distribuïda com de memòria compartida. Encara que tinguéssiu processadors i memòries d'un rendiment il·limitat, una xarxa d'interconnexió lenta faria que el rendiment dels programes paral·lel es degradés molt considerablement.

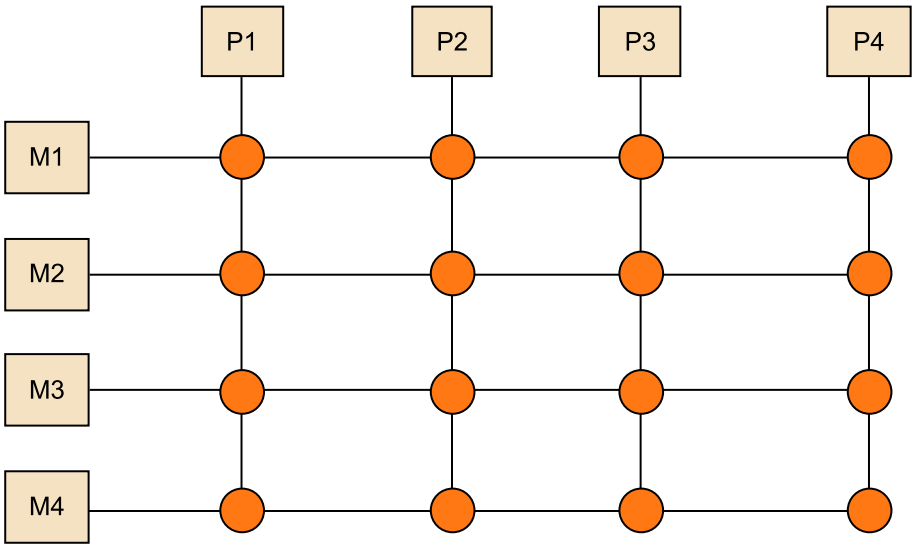
Tot i que algunes de les xarxes d'interconnexió tenen aspectes en comú, hi ha prou diferencies entre les xarxes d'interconnexió per a sistemes de memòria compartida i les de sistemes de memòria distribuïda

2.1. Xarxes de sistemes de memòria compartida

Actualment les dues xarxa d'interconnexió més utilitzades en sistemes de memòria compartida són els busos i *crossbars*. Cal recorda que un bus no és més que un conjunt de cables de comunicació en paral·lel juntament amb maquinari que controla l'accés al bus. La característica clau d'un bus és que els cables de comunicació són compartits per tots els dispositius que hi estan connectats. Els busos tenen l'avantatge de tenir un cos reduït i flexibilitat, i múltiples dispositius es poden connectar al bus amb molt poc cost addicional. Tot i així, com que els cables de comunicació són compartits, a mesura que el número de dispositius que utilitzen el bus creix, augmenten les possibilitats de que hi hagi contenció per la utilització del bus i, en conseqüència, el rendiment del mateix es redueix. Per tant, si connectem un número elevat de processadors a un bus, hem de suposar que aquests hauran d'esperar-se freqüentment per accedir a la memòria principal. Així doncs, a mesura que el tamany del sistema de memòria compartida augmenta, els busos són reemplaçats per xarxes d'interconnexió *switched*.

Les xarxes d'interconnexió *switched* utilitzen interruptors (*switches*) per controlar l'enrutament de les dades entre els dispositius que hi ha connectats. La figura 2 il·lustra un *crossbar* on les línies són links de comunicació bidireccional, els quadrats són nuclis o mòduls de memòria, i els cercles són interruptors.

Figura 2. *Crossbar* connectant 4 processadors i 4 memòries



Els interruptors individuals poden tenir una o dues configuracions tal i com mostra la figura 3. Amb aquests interruptors i almenys tants mòduls de memòria com processadors, només hi hauran conflictes entre dos nuclis intentant d'accedir a la memòria si els dos nuclis intenten accedir simultàniament al mateix mòdul de memòria. Per exemple, la figura 4 mostra la configuració dels interruptors si *P1* escriu a *M4*, *P2* llegeix de *M3*, *P3* llegeix de *M1* i *P4* escriu a *M2*.

Figura 3. Configuracions dels interruptors en un *crossbar*

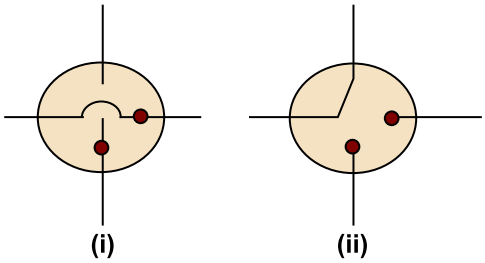
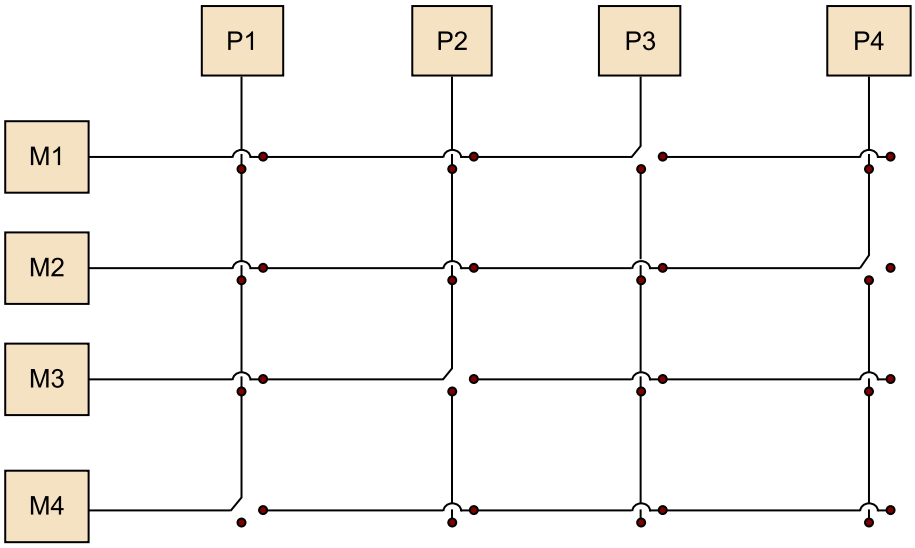


Figura 4. Accés a memòria per varis processadors simultàniament



Els *crossbars* permeten la comunicació simultània entre els diferents dispositius, per tant són molt més ràpids que els busos. En canvi, el cost dels interruptors i dels links és relativament alt. En general un sistema tipus bus és força més barat que un de basat en *crossbar* per a un sistema petit.

2.2. Xarxes de sistemes de memòria distribuïda

Les xarxes d'interconnexió de sistemes de memòria distribuïda normalment es divideixen en dos grups: interconnexions directes i interconnexions indirectes. En una interconnexió directa cadascun dels interruptors es connecta a un parell processador-memòria i els interruptors estan interconnectats entre ells. Les figures 5 i 6 mostren un anell i una graella toroïdal (*toroïdal mesh*) de dues dimensions, respectivament. Com anteriorment, els cercles són interruptors, els quadrats són processadors i les línies són links bidireccionals. Un anell és millor que un simple bus ja que permet múltiples connexions simultànies. En canvi, es pot veure clarament que hi haurà situacions en les quals alguns processadors hauran d'esperar hauran d'esperar a que d'altres acabin les seves comunicacions. La graella toroïdal és més cara que un anell perquè els interruptors són més complexes (han de tenir cinc links en canvi de tres de l'anell) i, si hi ha p processadors, el número de links és $3p$ en una graella toroïdal mentre que en un anell només seria de $2p$.

Figura 5. Anell

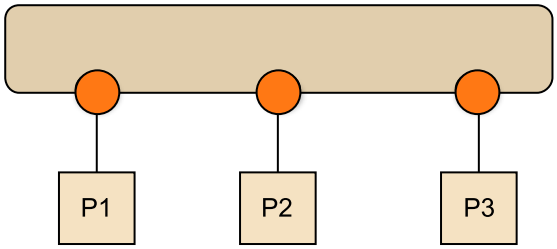
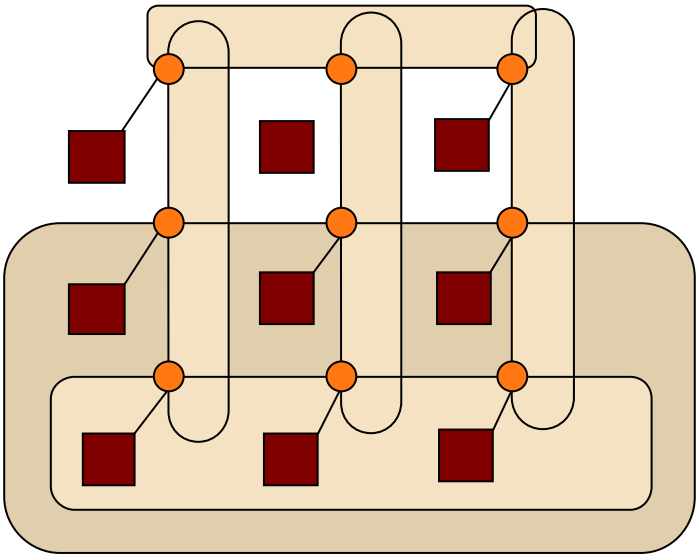
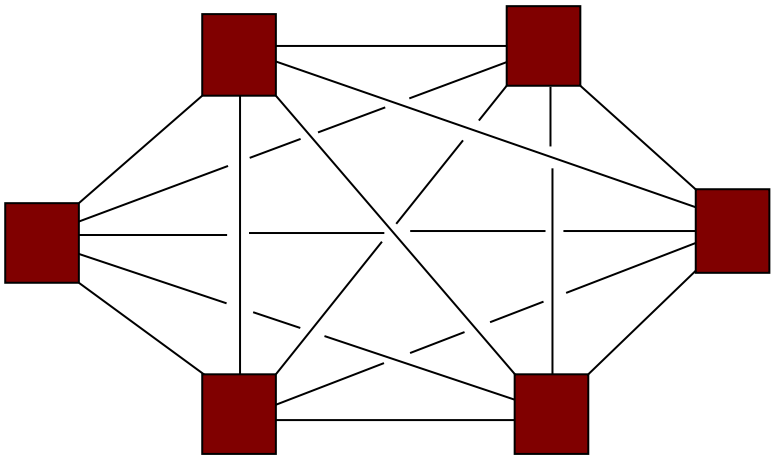


Figura 6. Graella toroïdal



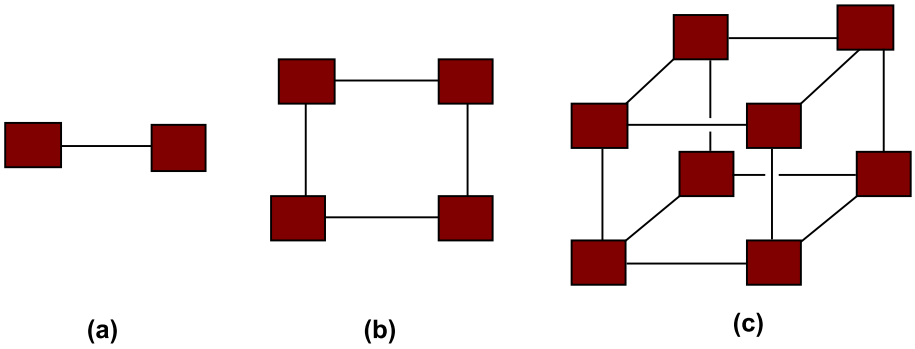
L'amplada de banda (*bandwidth*) d'un link és el rati al qual pot transmetre dades. Normalment es dóna en megabits or megabytes per segon. La xarxa d'interconnexió directa ideal és la completament connectada en la qual cada interruptor està connectat a tots els altres, tal com la figura 7. El problema d'aquest tipus de xarxa és que no es poden construir per a sistemes de més d'uns pocs nodes ja que requereixen un total de $p^2/p + p/2$ links, i cada interruptor s'ha de poder connectar a p links. Així doncs, es tracta més aviat d'una xarxa d'interconnexió teòrica del millor dels casos que no pas una de pràctica i s'utilitza com a referència per l'avaluació d'altres tipus de xarxa d'interconnexió.

Figura 7. Xarxa completament connectada



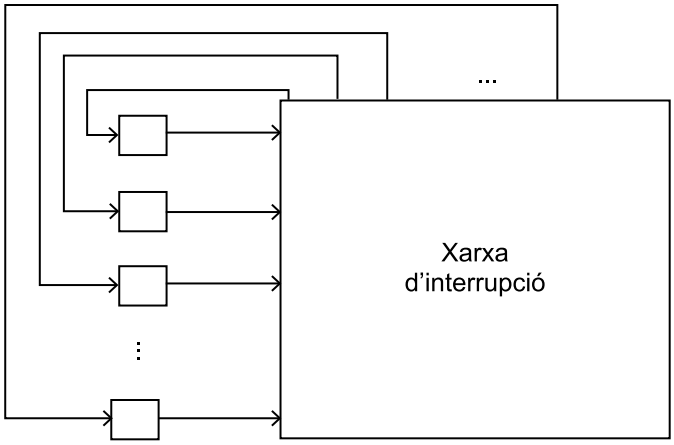
Un hipercub (*hypercube*) és una xarxa de connexió directa altament connectada que s'ha utilitzat en sistemes actuals. Els hipercubs es construeixen de forma inductiva: un hipercub d'una dimensió és un sistema completament connectat amb dos processadors. Un hipercub de dues dimensions es construeix a partir del d'una dimensió unint els corresponents interruptors, i un de tres dimensions es fa de la mateixa manera a partir d'un de dues dimensions tal i com mostra la figura 8. Per tant, un hipercub de dimensió d té $p=2^d$ nodes, i un interruptor en un hipercub de *d-dimensions* està connectat directament a un processador i d interruptors. Així doncs, un hipercub amb p nodes és més costós de construir que una graella toroïdal.

Figura 8. Hipercubs de (a) una, (b) dos, i (c) tres dimensions



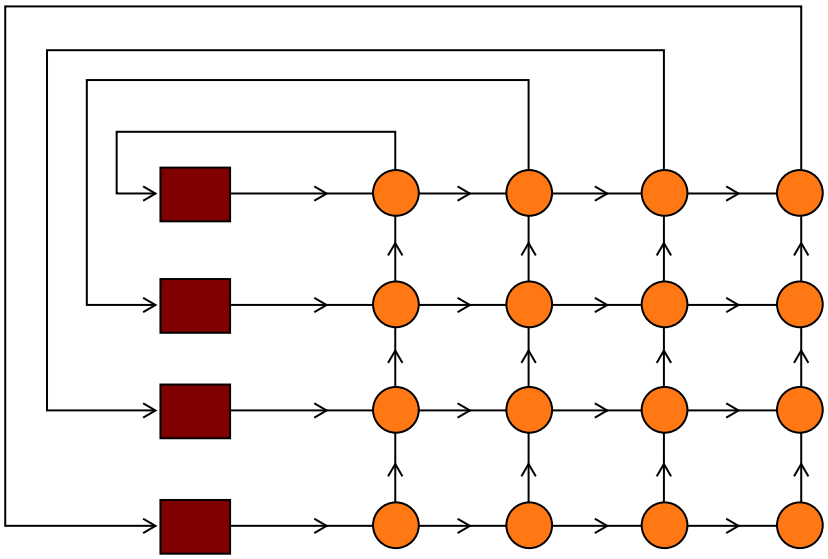
Les interconnexions indirectes són una alternativa a les connexions directes on els interruptors poden no estar connectats directament a un processador. Moltes vegades són vistos com a links unidireccionals i un conjunt de processadors, cadascun dels quals té un link d'entrada i una sortida, i una xarxa d'interrupció (*switching network*) tal i com mostra la figura 9.

Figura 9. Xarxa d'interrupció genèrica



El *crossbar* i la xarxa omega són exemples relativament simples de xarxes d'interconnexió indirectes. La figura 10 mostra el diagrama d'un *crossbar* de memòria distribuïda, el qual té links unidireccionals en contra del *crossbar* de memòria compartida que té links bidireccionals tal i com vam veure a la figura 4. Cal tenir en compte que mentre que dos processadors no intentin comunicar-se amb el mateix processador, tots els processadors poden comunicar-se simultàniament amb un altre processador.

Figura 10. Interconnexió amb *crossbar* de memòria distribuïda



En una xarxa omega com la mostrada en la figura 11, els interruptors són *crossbars* de dos-a-dos tal i com mostra la figura 12. Cal observar que al contrari del *crossbar*, hi ha comunicacions que no poden produir-se simultàniament. Per exemple, en la figura 11 si el processador 0 envia un missatge al processador 6, llavors el processador 1 no pot enviar un missatge al processador 7 simultània-

ment. Per l'altra banda, la xarxa omega és menys costosa que el *crossbar* ja que la xarxa omega utilitza $\frac{1}{2} p \cdot log_2(p)$ dels interruptors del *crossbars* 2x2 i, per tant, utilitza un total de $2p \cdot \log_2(p)$ interruptors, mentre que el *crossbar* n'utilitza p^2.

Figura 11. Xarxa omega

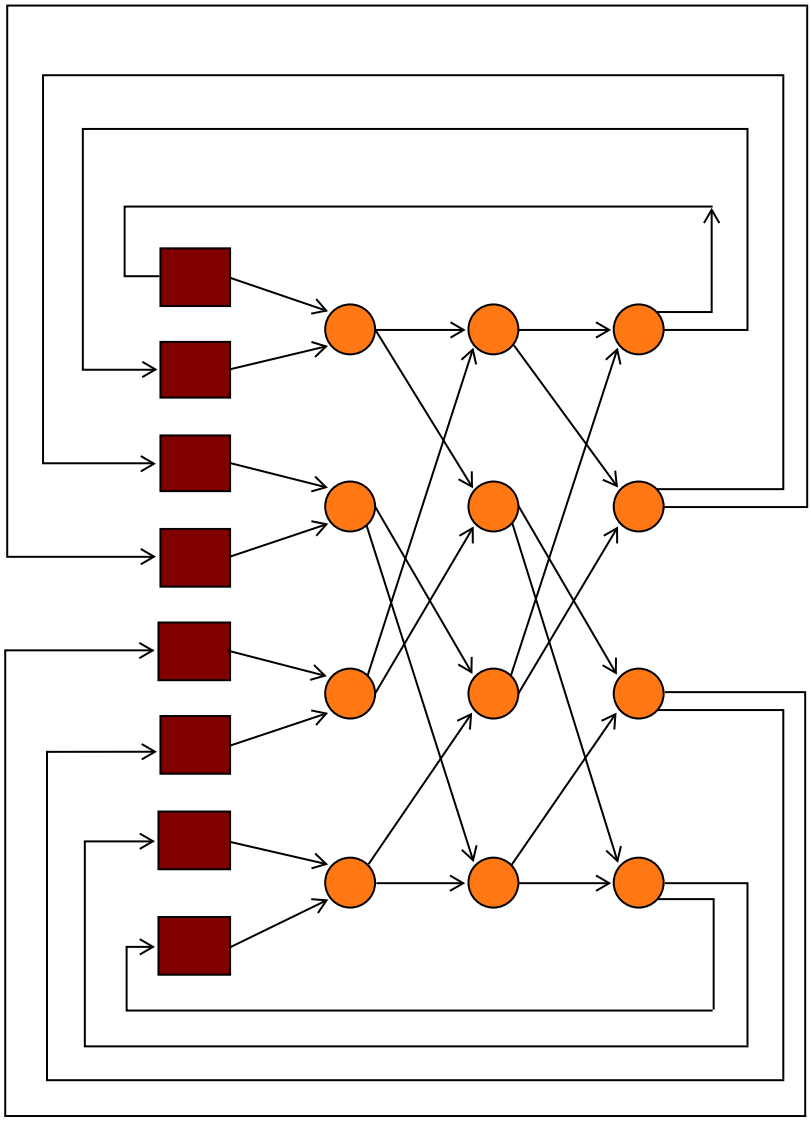
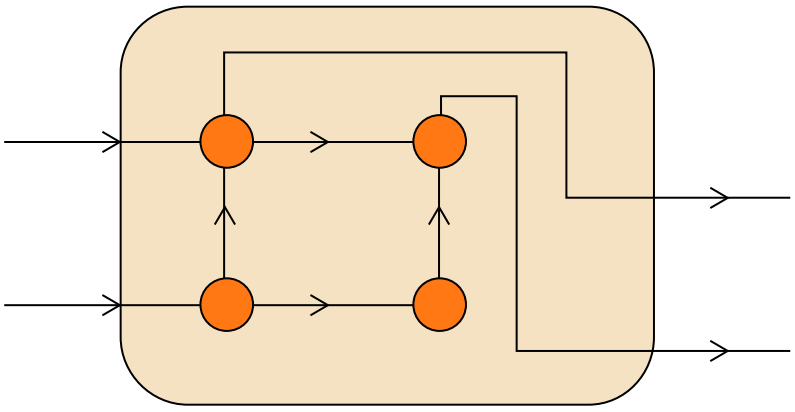


Figura 12. Interruptor d'una xarxa omega



2.3. Latència i amplada de banda

Sempre que hi ha dades que es transmeten estem interessats en saber quan trigaran les dades a arribar a la seva destinació. Això és així si parlem de transmetre dades entre la memòria principal i la memòria cau, memòria cau i registre, disc dur i memòria, o entre dos nodes en un sistema de memòria distribuïda o un de híbrid. Hi ha dos termes que s'acostumen a utilitzar per descriure el rendiment d'una xarxa d'interconnexió independentment del que connecti: la latència i l'amplada de banda. La latència és el temps que passa entre que l'origen comença a transmetre les dades i el destí comença a rebre el primer byte. L'amplada de banda és el rati al qual la destinació rep dades després de que hagi començat a rebre el primer byte. Per tant, si la latència d'una xarxa d'interconnexió és `l` segons i l'amplada de banda és `b` bytes per segon, llavors el temps que trigaria en transmetre un missatge de `n` bytes seria

```
Temps de transmissió d'un missatge = l + n/b
```

3. Sistemes d'arxius per a sistemes d'altes prestacions

En aquest capítol es presenten breument les característiques i qüestions bàsiques dels sistemes d'arxius que s'utilitzen en sistemes d'altes prestacions. També s'il·lustren alguns casos representatius d'aquest tipus de sistemes d'arxius, començant per NFS com a fonament dels sistemes d'arxius distribuït i continuant amb sistemes d'arxius paral·lels enfocats a les altes prestacions.

3.1. Sistemes d'arxius distribuïts

En general, un sistema d'arxius distribuït permet als processos l'accés transparent i eficient d'arxius que romanen en servidors remots. Les tasques principals són les d'organització, emmagatzematge, recuperació, compartiment i protecció dels arxius. També proporciona una interfície de programació que oculta als programadors els detalls de localització i com es du a terme realment l'emmagatzematge.

Alguns dels avantatges d'aquests sistemes d'arxius són, entre d'altres:

- Un usuari pot accedir als seus mateixos arxius des de diferents màquines.
- Diferents usuaris poden accedir als mateixos arxius des de diferents màquines.
- És més fàcil d'administrar, ja que només hi ha un servidor o grup de servidors.
- Es millora la fiabilitat, ja que es pot afegir, per exemple, tecnologia RAID (*redundant array of independent disks*).

Alhora, hi ha alguns reptes i qüestions importants que aquests tipus de sistemes d'arxius intenten solucionar. Per exemple:

- Escalabilitat: el servei s'ha de poder estendre incrementalment per a gestionar un rang ampli de càrregues i mides de xarxa.
- Tolerància a fallades: clients i servidors han d'operar correctament davant de fallades.
- Consistència: diferents clients han de veure el mateix directori i contingut dels arxius si hi accedeixen al mateix temps.
- Seguretat: mecanismes de control d'accés i autenticació.

- Eficiència: la seva utilització ha de ser similar al sistemes de fitxers locals.

3.1.1. *Network file system* (NFS)

NFS és un sistema de compartició d'arxius entre màquines d'una xarxa de tal manera que tenim la impressió de treballar en el nostre disc dur local. Un sistema Linux pot treballar com a servidor o com a client d'NFS (o com tots dos), la qual cosa implica que pot exportar sistemes d'arxius a altres sistemes i muntar els sistemes d'arxius que altres màquines exporten.

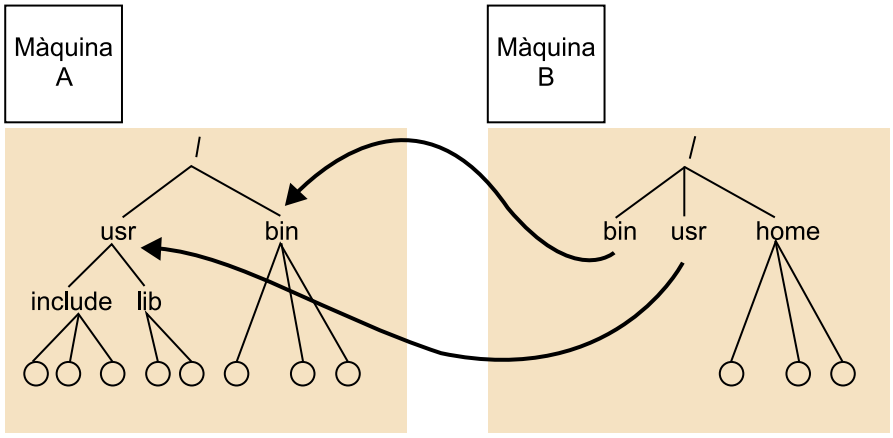
Per a suportar NFS, Linux utilitza una combinació de suport de nucli i dimonis en execució contínua per a proporcionar la compartició d'arxius NFS. Així doncs, el suport NFS ha d'estar actiu en el nucli del sistema operatiu. Per una altra banda, NFS utilitza RPC (*remote procedure calls*) per a encaminar peticions entre clients i servidor. L'ús d'RPC implica que el servei *portmap* ha d'estar disponible i actiu. El *portmap*, és un dimoni encarregat de crear connexions RPC amb el servidor i de permetre-hi l'accés o no.

Nota

Estudiarem RPC en el mòdul didàctic enfocats a sistemes distribuïts per altes prestacions.

La figura 13 mostra un exemple d'utilització d'NFS mitjançant el muntatge de directoris remots per a compartir arxius.

Figura 13. Exemple d'utilització d'NFS per a compartir arxius mitjançant el muntatge de directoris remots.



En l'exemple, la màquina A exporta els directoris /usr i /bin, i en la màquina B es munten per a la seva utilització.

Els principals processos RPC necessaris són els següents, tot i que n'hi intervenen més:

- *rpc.mountd*: és un procés que rep la petició de muntatge des d'un client NFS i comprova si coincideix amb un sistema d'arxius actualment exportat.

- *rpc.nfsd*: és un procés que implementa els components de l'espai de l'usuari del servei NFS; treballa amb el nucli Linux per satisfer les demandes dinàmiques de clients NFS, com ara proporcionar processos addicionals del servidor perquè els clients NFS l'utilitzin.

La configuració del servidor NFS, la qual es fa mitjançant el fitxer /etc/exports, permet especificar qüestions molt importants relacionades amb la compartició d'arxius. Entre aquestes, es troba la llista de màquines autoritzades en la compartició juntament amb certes opcions com, per exemple:

- *ro*: només lectura (*read-only*). Les màquines no poden canviar el sistema de fitxers.
- *rw*: lectura-escriptura (*read-write*).
- *async*: permet al servidor escriure les dades en el disc quan ho cregui convenient.
- *sync*: totes les escriptures al disc s'han de fer abans de tornar el control al client.
- *wdelay*: el servidor NFS retarda l'escriptura en disc sospita que una altra petició d'escriptura és imminent. *no_wdelay*: per a desactivar aquesta opció, la qual només funciona si useu l'opció sync.
- *root_squash*: proporciona als usuaris privilegiats (*root*) connectats remotament privilegis com a *root* local. *no_root_squash*: el desactiva.
- *all_squash*: per a reconvertir a tots els usuaris.

També es poden especificar grups de xarxes mitjançant grups de noms de domini o adreces IP.

Activitat

Busqueu informació del funcionament intern d'NFS i penseu com faríeu la vostra pròpia implementació d'un servidor/client d'NFS.

3.2. Sistemes d'arxius paral·lels

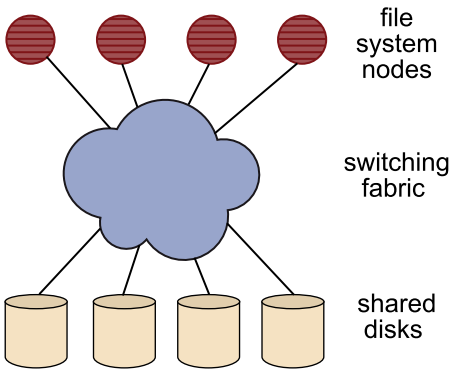
Els sistemes d'arxius paral·lels intenten donar solució a la necessitat d'entrada/sortida de certes aplicacions que gestionen volums massius de dades i, per tant, requereixen un nombre enorme d'operacions d'entrada/sortida sobre els dispositius d'emmagatzematge. De fet, en els darrers anys s'ha vist un creixement molt significatiu de la capacitat dels discos, en canvi no ha estat igual en les seves prestacions com ara amplada de banda i latència. Això ha produït un desequilibri important entre capacitat de processament i entrada/sortida que fa que afecti molt negativament les aplicacions que són força intensives en operacions d'entrada/sortida.

Els sistemes d'arxius paral·lels es basen en el mateix principi que el còmput per millorar les prestacions, és a dir, proporcionar entrada/sortida paral·lela. Això fa que la distribució de dades sigui entre múltiples dispositius d'emmagatzematge i que es pugui accedir a les dades en paral·lel. Hi ha diferents tipus de connexió de dispositius d'emmagatzematge:

- DAS (*direct-attached storage*): solució "clàssica" de disc associat a node.
- NAS (*network-attached storage*): node que gestiona un conjunt de discos.
- SAN (*storage area networks*): xarxa dedicada a l'emmagatzematge.
  - Emmagatzematge no vinculat a cap node (discos de xarxa).
  - Xarxes de comunicació separades per a dades d'aplicació i fitxers.
  - Xarxes d'emmagatzematge que inclouen *hubs*, *switches*, etc. La tecnologia més utilitzada és *fibre channel*.
  - Connectivitat total entre nodes i dispositius: qualsevol node pot accedir directament a qualsevol dispositiu.
  - Connectivitat directa entre dispositius: per exemple per a fer còpies entre dispositius (agilitza còpies de seguretat, replicació, etc.).

Respecte a l'organització dels sistemes d'arxius paral·lels, aquests fan servir tècniques de *stripping*, la qual consisteix a emmagatzemar les dades d'arxiu distribuïdes entre discos del sistema de manera similar al RAID-0 però per programari i entre diversos nodes. Els sistemes d'arxius es comparteixen a partir d'un repartiment funcional en dos nivells. El nivell inferior és el servei d'emmagatzematge distribuït (per exemple, SAN), i el superior és el sistema d'arxius a cada node de còmput. Per a accedir als discos com si fossin locals), cada node de còmput ha de gestionar metainformació de les dades. La figura 14 il·lustra l'organització conceptual en capes dels sistemes d'arxius paral·lels.

Figura 14. Organització funcional per capes d'un sistema d'arxius paral·lel



Alguns exemples de sistemes d'arxius paral·lels són GPFS, Lustre, PVFS o Google File System. A continuació es descriuen les característiques bàsiques del dos primers a tall d'exemple.

3.2.1. General Parallel File System (GPFS)

GPFS va ser desenvolupat per IBM, i com a sistema d'arxius paral·lel permet als usuaris compartir l'accés a dades que estan disperses en múltiples nodes; permet la interacció per mitjà de les interfícies estàndard d'UNIX.

GPFS permet millorar el rendiment del sistema i les seves característiques principals són:

- Garanteix la consistència de les dades.
- Té una alta recuperabilitat i disponibilitat de les dades.
- Proporciona una alta flexibilitat al sistema.
- Administració simplificada.

La millora del rendiment és deguda a factors com els següents:

- Permet que múltiples processos o aplicacions accedeixin simultàniament a les dades des de tots els nodes utilitzant crides estàndard del sistema.
- Increment de l'amplada de banda de cadascun dels nodes que intervé en el sistema GPFS.
- Balanceja la càrrega uniformement entre tots els nodes del sistema GPFS. Un disc no pot tenir més activitat que un altre.
- Suporta dades de grans dimensions.
- Permet lectures i escriptures concurrents des de qualsevol node del sistema GPFS.

GPFS utilitza un sofisticat sistema d'administració que garanteix la consistència de les dades alhora que permet rutes múltiples i independents per a arxius amb el mateix nom, des de qualsevol lloc del clúster. Quan la càrrega de certs nodes és molt alta, el GPFS pot trobar una ruta alternativa pel sistema d'arxius de dades.

Per tal de recuperar i per facilitar la disponibilitat de les dades, el GPFS crea registres *logs* separats per a cadascun dels nodes que intervé en el sistema. El GPFS permet que s'organitzi el maquinari dins d'un nombre de grup de falla. A més, la funció de replicació del GPFS permet determinar quantes còpies dels

arxius cal mantenir. El GPFS també permet afegir recursos, tant si són discos com nodes dinàmicament sense necessitat d'aturar i tornar a posar en marxa el sistema.

En un sistema GPFS cadascun dels seu nodes s'organitza a partir dels components següents:

- Tecles d'ordre de gestió/administració.
- Extensions del nucli.
- Un dimoni multifil.
- Una capa portable de codi obert.

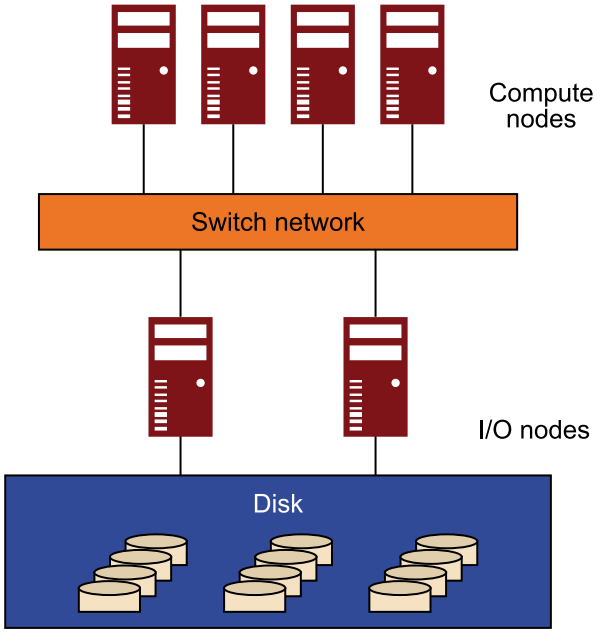
L'extensió del nucli proporciona una interfície entre el sistema operatiu i el sistema GPFS, la qual cosa facilita la manipulació de les dades en un entorn GPFS, ja que amb les tecles d'ordre del sistema operatiu es pot fer qualsevol operació sobre el sistema GPFS. Per exemple, per a copiar un arxiu només cal executar la sintaxi la tecla d'ordre habitual cp fitxer.txt prova.txt.

Els dimonis del GPFS s'executen en tots els nodes d'entrada/sortida i un administrador de *buffer* per GPFS. Totes les entrades/sortides estan protegides per un administrador de *token*, el qual assegura que el sistema d'arxius en múltiples nodes compleixi amb l'atomicitat i proporciona la consistència de dades dels sistemes d'arxius. Els dimonis són processos multifil amb alguns fils dedicats a processos específics. Això assegura que el servei no es vegi interromput perquè un altre fil estigui ocupat amb una rutina de xarxa. Les funcions específiques d'un dimoni són:

- Assignació d'espai en disc per a nous arxius.
- Administració de directoris.
- Assignació de bloqueig per a la protecció de la integritat de les dades i de les metadades.
- Els servidors de discos són iniciats amb un fil del dimoni.
- La seguretat també s'administra per al dimoni en conjunt amb l'administrador dels sistemes de fitxers.

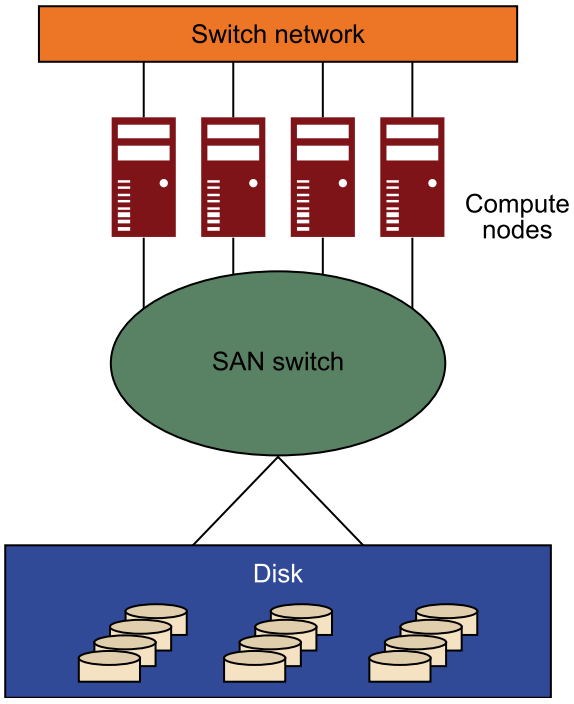
Les figures 15 i 16 il·lustren sistemes GPFS amb la utilització de nodes dedicats a entrada/sortida respectivament i utilitzant infraestructura SAN.

Figura 15. Configuració d'un sistema GPFS mitjançant nodes específics d'entrada/sortida



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

Figura 16. Configuració d'un sistema GPFS mitjançant SAN



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

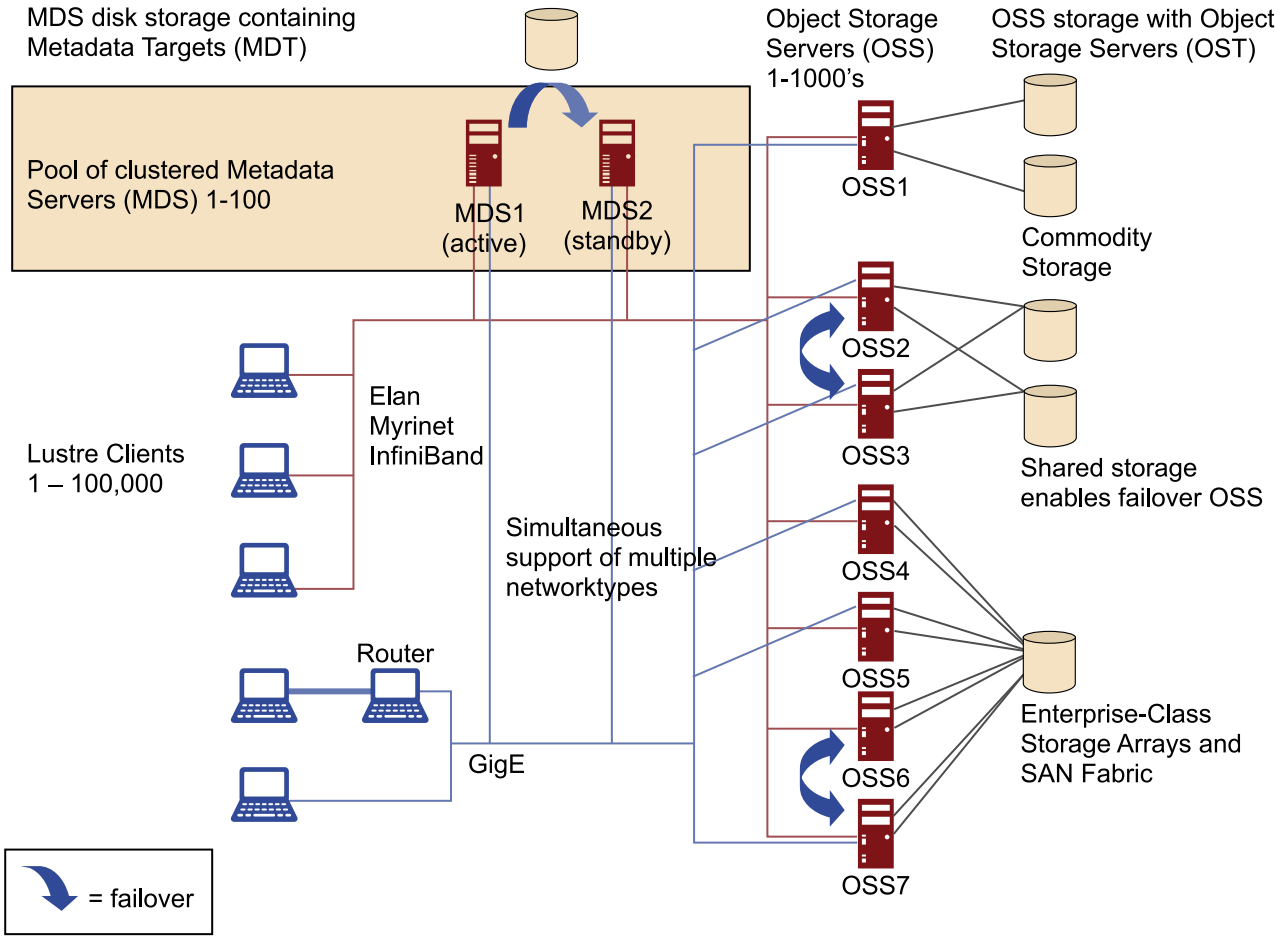
3.2.2. Linux Cluster File System (Lustre)

Lustre és un sistema d'arxius distribuït de codi obert, normalment utilitzat en clústers a gran escala. El nom és una barreja de *Linux* i *clústers*. El projecte intenta proporcionar un sistema d'arxius per a clústers de desenes de milers de nodes amb petabytes de capacitat d'emmagatzematge, sense comprometre la velocitat o la seguretat. Els clústers Lustre contenen tres tipus de components:

- Clients que accedeixen al sistema.
- *Object storage servers* (OSS) que proporcionen el servei d'entrada/sortida d'arxius.
- *Metadata servers* (MDS) que gestionen els noms i els directoris del sistema d'arxius.

La figura 17 mostra l'estructura i organització d'un clúster Lustre.

Figura 17. Estructura d'un clúster Lustre a escala (font:www.lustre.org)



L'emmagatzematge dels servidors és en particions i opcionalment organitzat mitjançant un volum lògic de gestió (LVM) i formatat com un sistema de fitxers. Els servidors OSS i l'MDS de Lustre llegeixen, escriuen i modifiquen dades en el format imposat per aquest sistema. Cada OSS pot responsabilitzar-se de múltiples OST, un per cada volum, i el trànsit d'entrada/sortida es balanceja. Depenent del maquinari dels servidors un OSS normalment s'encarrega de dos a vint-i-cinc objectius, cada objectiu aproximadament de 8 terabytes de capacitat. La capacitat del sistema Lustre és la suma de les capacitats proporcionades pels objectius. Un OSS ha de balancejar l'amplada de banda del sistema per a evitar possibles colls d'ampolla. Per exemple, seixanta-quatre servidors OSS, cadascun amb dos objectius de 8TB, proporcionen un sistema d'arxius d'una capacitat propera a 1 PB. Si el sistema utilitza 16 discos SATA d'1

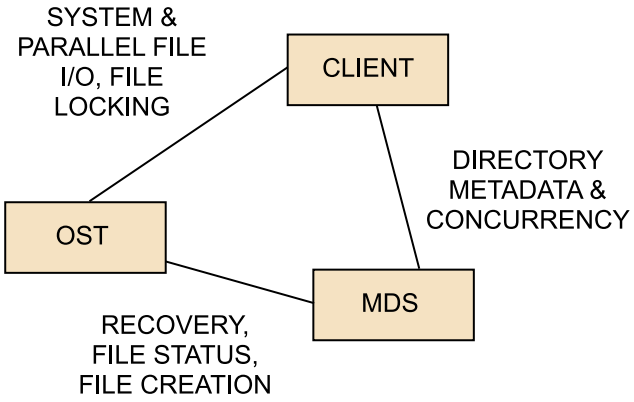
terabyte, és possible aconseguir 50 MB/sec per cada dispositiu, i proporcionar 800 MB/sec d'amplada de banda. Si aquest sistema s'utilitzés com a *back-end* d'emmagatzematge amb un sistema de xarxa, com ara InfiniBand, que suporta una amplada de banda similar, llavors cada OSS podria proporcionar 800 MB/sec al *throughput* d'entrada/sortida.

Normalment els servidors OSS no utilitzen dispositius interns, però utilitzen una matriu d'emmagatzematge sobre *fiber channel* o connexions SCSI (SAS), com en el cas del GPFS. En qualsevol cas, tant el *software* com el maquinari RAID segueixen el format de *striping* de RAID 5 o 6. La memòria dels OSS s'utilitza com a memòria cau en les lectures d'arxius i en alguns casos per a escriure dades. La utilització de la CPU és mínima quan s'utilitza RDMA.

L'emmagatzematge en Lustre només es presenta en els nodes servidor, mai en els clients. Si es vol utilitzar un sistema més resistent als errors, es pot emmagatzemar la informació en múltiples servidors. En tot cas, l'ús de SAN amb *switches* cars es pot evitar, ja que les connexions punt a punt entre els servidors i *arrays* d'emmagatzematge són la millor elecció. Per als nodes MDS es manté la mateixa consideració. Les metadades han d'ocupar entre l'1% i el 2% de la capacitat del sistema, però l'accés a les dades per a l'emmagatzematge MDS és lleugerament diferent de l'emmagatzematge OSS: mentre al primer s'accedeix mitjançant metadades, amb nombroses cerques i operacions de lectura i escriptura de petites dades, al segon s'accedeix amb un patró d'entrada/sortida que envolta grans transferències. Els sistemes Lustre són força senzills de configurar.

Els clients Lustre utilitzen el sistema i interactuen amb els *object storage targets* (OSTs) per a la lectura/escriptura d'arxius i amb els *metadata servers* (MDS). Tant MDS com OST i OSS es poden trobar en el mateix node o en nodes diferents. La figura 18 ens presenta el diàleg entre els diferents elements.

Figura 18. Diàleg entre els components de Lustre



Un OST proporciona emmagatzematge dels objectes de les dades (*chunks*). Els objectes es presenten com a *inodes* i l'accés a aquests objectes és proporcionat pels OST que duen a terme el servei d'entrada/sortida al clúster Lustre. Els noms són tractats per les metadades que gestionen els *inodes*. Els *inodes* es po-

den presentar com a directoris, com a enllaços simbòlics o dispositius especials i la seva informació i les seves metadades s'emmagatzemen en els MDS. Quan un *inode* de Lustre representa un arxiu, les metadades fan referència als objectes emmagatzemats en els OST. En el disseny d'un sistema Lustre és fonamental que els OST assignin la informació en blocs facilitant la distribució i l'escalabilitat de les metadades. De fet, els OST reforcen la seguretat respecte a l'accés dels objectes.

Un MDS gestiona totes les operacions referents a un arxiu com poden ser assignar o actualitzar referències. L'MDT proporciona emmagatzematge per a les metadades del sistema. Les metadades gestionades consisteixen en arxius que contenen els atributs de les dades emmagatzemades en els OST.

El servidor de gestió (MGS) defineix informació sobre la configuració de tots els components presents. Els elements de Lustre contacten amb aquest per proporcionar informació, mentre que els clients ho fan per rebre-la. L'MGS pot proporcionar actualitzacions en la configuració dels elements i dels clients. L'MGS necessita el seu propi disc d'emmagatzematge, però es pot compartir un disc amb un MDT. Un MGS no es considera una part d'un únic sistema sinó que pot proporcionar mecanismes de configuració per a altres components Lustre.

Els clients són els usuaris del sistema de fitxers. Normalment se'ns presenten com a nodes computacionals o de visualització. Els clients de Lustre necessiten el programari Lustre per a muntar el sistema de fitxers, ja que Lustre no és NFS. El client no és més que un programari que consisteix en una interfície entre el sistema d'arxius virtual del Linux i el del mateix Lustre.

Els servidors i clients es comuniquen els uns amb els altres mitjançant una API de xarxa coneguda com a Lustre Networking (Telnet). Telnet interacciona amb una gran varietat de xarxes. Telnet proporciona les decisions i els esdeveniments per a cada missatge d'una connexió de xarxa suportant *routing* entre nodes situats en diferents xarxes.

Lustre s'utilitza per a emmagatzemar dades, per al qual cosa es requereix un cert grau de redundància per a assegurar-ne la fiabilitat. La primera idea per a combatre la pèrdua d'informació és utilitzar OST redundants en forma de mirall. Aquesta implementació consisteix a utilitzar un OST que emmagatzemarà una rèplica exacta de la informació, de manera que si el primer OST fallés, els objectes romandrien en un altre OST. Una altra característica d'aquesta funcionalitat és la possibilitat de balancejar la càrrega, ja que, en disposar de múltiples còpies de la mateixa informació, la càrrega de lectura pot ser compartida per tots dos OST.

Lustre proporciona suport a la recuperació d'un node quan es presenta una fallada. Quan Lustre es troba en mode de recuperació, tots els seus servidors (MDS, OSS) passen a un estat de bloqueig, és a dir, la informació que no ha

estat desada es manté en la memòria cau del client. Per emmagatzemar aquesta informació, el sistema reinicia en mode de recuperació i fa que els clients l'escriguin en el disc. En mode de recuperació, els servidors tracten de contactar amb tots els clients i de contestar a les seves peticions. Si tots els clients han estat contactats i són recuperables (no han estat reiniciats), llavors es procedeix a la recuperació i el sistema emmagatzema les dades que es troben en la memòria cau dels clients. Si un o més d'un client no és capaç de tornar a connectar (a causa de fallades de maquinari o reinici del client), llavors el procés de recuperació caduca, la qual cosa fa que els clients siguin expulsats. En aquest cas, si hi ha informació en la memòria cau del client que no ha estat desada, aquesta no s'emmagatzemarà en el disc i es perdrà.

4. Sistemes de gestió de cues i planificació

Sistemes d'altes prestacions actuals

Les característiques d'aquests sistemes són la latència reduïda i la gran amplada de banda de la xarxa d'interconnexió.

Els sistemes d'altes prestacions actuals són principalment clústers de computadors interconnectats amb xarxes d'altes prestacions. En aquests sistemes normalment hi ha múltiples usuaris que volen accedir als recursos per tal d'executar programes (paral·lels) desenvolupats mitjançant models de programació com els que hem vist anteriorment (per exemple, MPI). Per a organitzar l'accés dels diferents usuaris als recursos i gestionar aquests eficientment (per exemple, per a maximitzar-ne la utilització), els sistemes d'altres prestacions disposen de sistemes de gestió de recursos basats en cues.

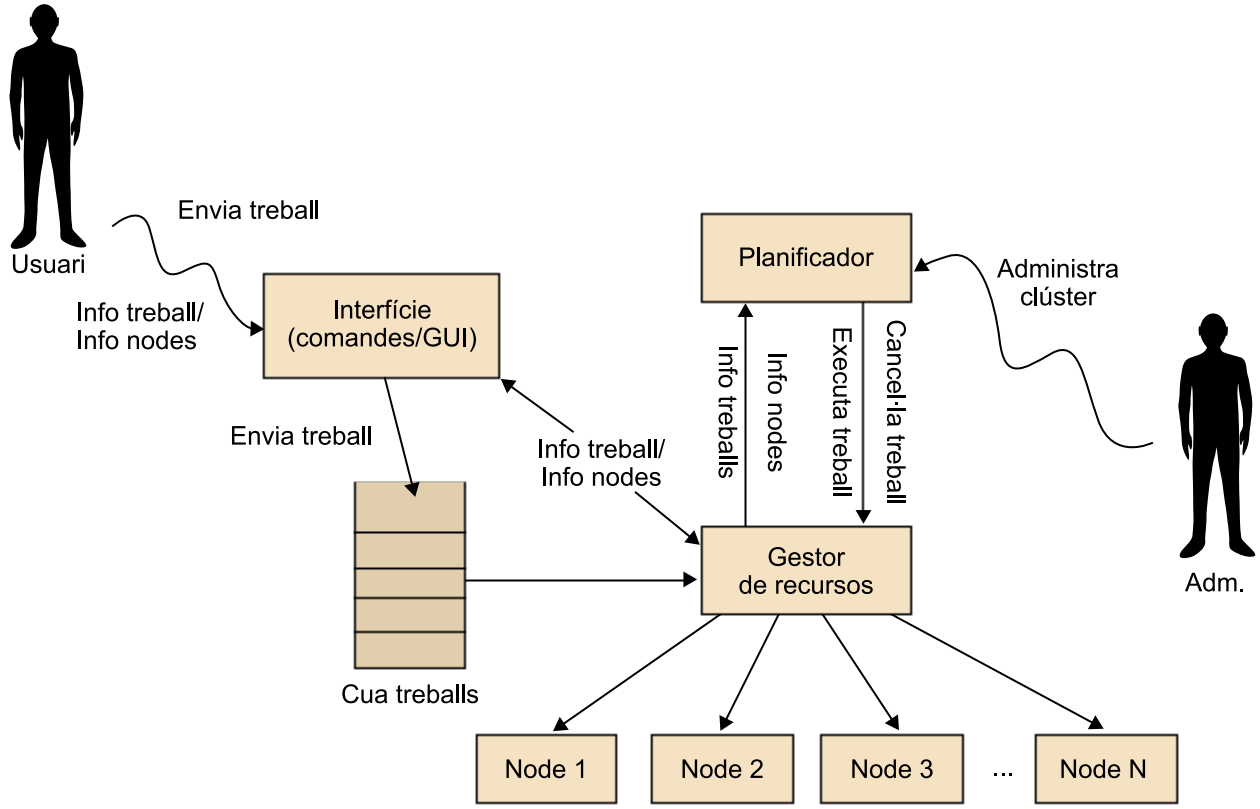
Exemples de conflictes

Alguns usuaris podrien fer un ús desmesurat dels recursos o fins i tot processos erronis podrien malgastar els recursos sense que es poguessin controlar.

De fet, en sistemes d'altes prestacions no té sentit proporcionar als usuaris accés als recursos de manera interactiva, ja que seria un caos: hi hauria conflictes en la utilització dels recursos, poca eficiència en la utilització i compartició de recursos. A més, els usuaris no s'han de preocupar de problemes relacionats amb la gestió dels computadors, com ara el balanceig de la càrrega. Així doncs, els sistemes de computació d'altes prestacions es basen en aplicacions que es poden emmagatzemar en una cua i executar posteriorment en segon pla. Aquests sistemes s'anomenen tradicionalment *sistemes per lot* (*batch*).

La figura 19 mostra un esquema d'un clúster d'altes prestacions amb els components essencials per a la seva gestió. També cal tenir en compte que en els sistemes d'altes prestacions normalment hi ha el gran gruix de nodes que s'encarreguen de la computació, però normalment no s'hi pot accedir directament per mesures de seguretat i gestió, i hi ha un conjunt de nodes (que poden ser de les mateixes característiques que la resta de nodes o diferents) que s'utilitzen com a porta d'entrada (*login nodes*). Normalment, aquests nodes d'accés permeten desenvolupar les aplicacions i disposen de sistemes complets amb eines de compilació, depuració, etc. i permeten als usuaris enviar els seus treballs als sistemes de cues per a la seva execució.

Figura 19. Esquema d'un sistema clúster amb els components de gestió de recursos



4.1. Sistemes de gestió de cues

Els sistemes de gestió de cues són sistemes d'administració, planificació i execució de treballs que s'envien a un clúster. A continuació s'exposen breument els sistemes de gestió de cues PBS, que actualment és el més habitual en la computació d'altes prestacions.

4.1.1. PBS (*portable batch system*)

El PBS[1] és un sistema de cues dissenyat originalment per la NASA que ha arribat a ser uns dels sistemes de cues més populars en la computació d'altes prestacions. De fet, un gran percentatge de sistemes d'altes prestacions actuals utilitzen l PBS com a sistema de cues.

(1)De l'anglès *portable batch system*.

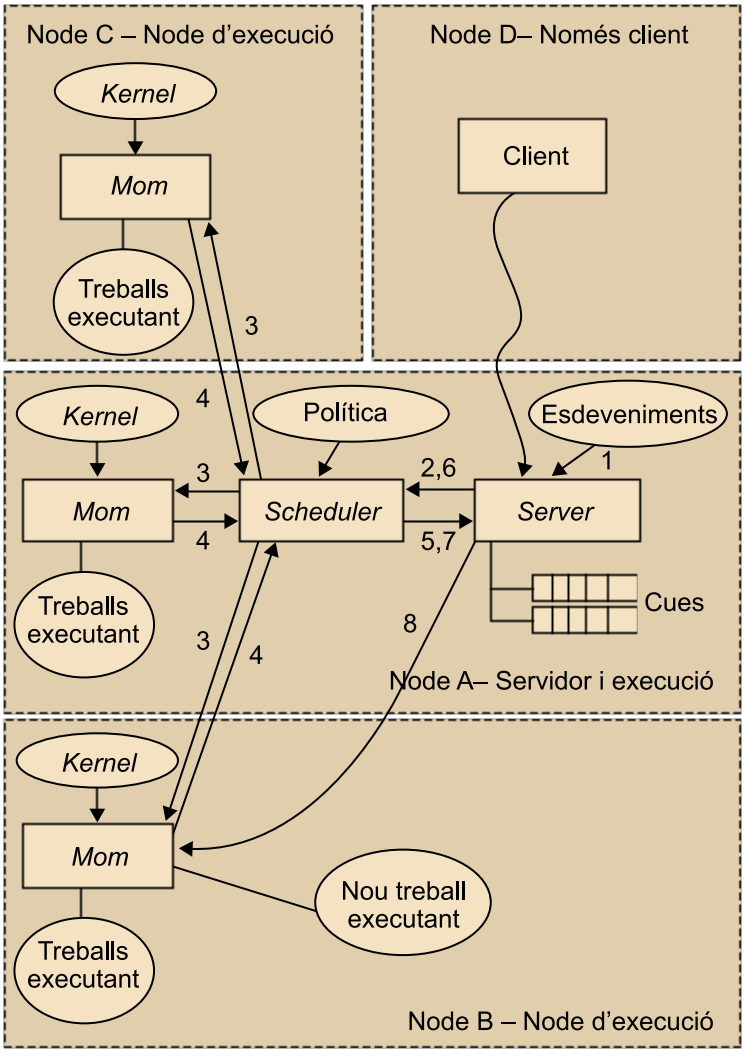
El PBS proporciona una sèrie d'eines per a la gestió de treballs per lots (*batch*), i utilitza una unitat de programació de tasques. A més, permet la planificació de treballs a diferents computadors i també definir i implementar polítiques sobre la utilització d'aquests recursos.

La figura 20 mostra l'esquema del sistema de cues, en què hi podem diferenciar els components que el formen i les interaccions entre aquests. Podem diferenciar clarament tres components bàsics:

- *server*. És un dimoni encarregat de rebre els treballs a executar i d'esperar les ordres de l'usuari.

- *mom* (o *job executor*). És un dimoni que s'executa en cadascun del nodes de còmput i que envia i rep els treballs a executar.
- *scheduler*. És un planificador que decideix quins treballs s'han d'executar en funció de la política que estigui establerta.

Figura 20. Esquema del sistema de cues PBS



Aquests components interactuen mitjançant els passos que es descriuen a continuació d'una manera cíclica:

- Un *event* diu al *server* que comenci un cicle de planificació.
- El *server* envia una petició de planificació a l'*scheduler*.
- L'*scheduler* sol·licita informació sobre els recursos als *mom*.
- Els *mom* retornen la informació sol·licitada a l'*scheduler*.
- L'*scheduler* sol·licita informació sobre el treball al *server*.
- El *server* retorna la informació sobre l'estat del treball a l'*scheduler* i pren la decisió d'executar el treball o no en funció de la política establerta.
- L'*scheduler* envia una petició d'execució al *server*.
- El *server* envia el treball al *mom* per a ser executat.

La manera en què els usuaris interactuen amb el sistema de cues és amb d'un seguit d'instruccions que es poden cridar des del sistema operatiu.

Per a executar un programa l'usuari ha de definir un treball mitjançant un fitxer *script*. A més d'indicar quin és el programa a executar i els arguments, en aquest fitxer l'usuari permet:

- Especificar si és un treball per lots o interactiu. En el cas de ser interactiu no vol dir que s'executi immediatament, sinó que quan s'executa permet a l'usuari d'interactuar amb el programa a diferència dels treballs per lots, que s'executen en segon pla sense que l'usuari hi pugui interactuar directament.

- Definir una llista amb els recursos necessaris.

- Definir la prioritat del treball per a la seva execució.

- Definir el temps d'execució, que és una informació important per a poder planificar els treballs eficientment.

- Especificar si es vol enviar un correu electrònic a l'usuari quan l'execució comença, acaba o és avortada.

- Definir dependències.

- Sincronitzar el treball amb altres treballs.

- Especificar si es vol fer *checkpoiting* (en cas que el sistema operatiu ofereixi aquesta possibilitat). *Checkpointing* és especialment important en execucions molt llargues en què es vol poder monitoritzar l'aplicació durant el temps d'execució o bé per qüestions de seguretat contra possibles fallades.

La llista de recursos que es poden especificar en un fitxer de definició d'un treball permeten, a més de garantir que el treball es podrà executar, millorar la utilització del sistema, ja que el planificador pot prendre decisions més acurades. Una llista amb els recursos més comuns (no completa) es descriu a continuació:

Decisions acurades

Un exemple seria no deixar nuclis sense feina perquè no hi ha prou memòria en un node per a assignar-hi un altre treball.

- *cput*. Temps màxim de CPU usat per tots els processos del treball.
- *pcput*. Temps màxim de CPU usat per cadascun dels processos del treball.
- *mem*. Quantitat màxima de memòria física utilitzada per a un treball.
- *pmem*. Quantitat màxima de memòria física utilitzada per cadascun del processos d'un treball.
- *vmem*. Quantitat màxima de memòria virtual utilitzada per un treball.
- *pvmem*. Quantitat màxima de memòria virtual utilitzada per cadascun del processos d'un treball.
- *walltime*. Temps d'execució (de rellotge, no de CPU).

- *file*. Mida màxima de qualsevol fitxer que pot crear el treball.
- *host*. Nom dels nodes computacionals en què s'executa el treball.
- *nodes*. Nombre i/o tipus de nodes a reservar per a l'ús exclusiu del treball.

El PBS proporciona una interfície per a l'intèrpret d'ordres i una interfície gràfica. Tot i així el més habitual en la computació d'altes prestacions és utilitzar la interfície per mitjà de l'intèrpret d'ordres per a enviar, monitoritzar, modificar i eliminar treballs. Hi ha diferents tipus d'ordres en funció del tipus d'usuari que les fa servir:

- Ordres d'usuari: `qsub`, `qstat`, `qdel`, `qselect`, `qrerun`, `qorder`, `qmove`, `qhold`, `qalter`, `qmsg`, `qrls`.
- Ordres d'operació: `qenable`, `qdisable`, `qrun`, `qstart`, `qstop`, `qterm`.
- Ordres d'administració: `qmgr`, `pbsnodes`.

A continuació veurem la sintaxi d'algunes de les ordres més utilitzades des del punt de vista de l'usuari 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]
```

L'ordre `qsub` envia un nou treball al servidor del PBS per a la seva execució. Per defecte es considera que el treball es posarà en la cua i s'executarà en segon pla, però també es permet executar el treball (igualment quan planificar ho considerem oportú) de manera interactiva, si s'especifica l'opció `-q`. Típicament, el `script` que s'envia per mitjà de l'ordre és un *shell script* de l'intèrpret d'ordres com ara `sh` o `csh`.

Les opcions de l'ordre `qsub` que s'indiquen anteriorment permeten especificar atributs que afectaran el comportament del treball. A més, l'ordre `qsub` passarà al treball una llista de variables d'entorn que estaran disponibles durant l'execució del treball. El valor de les variables `HOME`, `LANG`, `LOGNAME`, `PATH`, `MAIL`, `SHELL` i `TZ` es pren de les variables d'entorn de l'ordre `qsub`. Aquests valors s'assignaran a variables que comencen per "`PBS_O_`". Per exemple, el treball tindrà accés a una variable d'entorn anomenada `PBS_O_HOME`, que tindrà el valor de la variable `HOME` en l'entorn de `qsub`. A part de les variables d'entorn anteriors, també hi ha les següents disponibles per al treball:

- `PBS_O_HOST`. És el nom del servidor en què l'ordre `qsub` s'està executant.
- `PBS_O_QUEUE`. És el nom de la cua inicial a la qual es va enviar el treball.
- `PBS_O_WORKDIR`. És la ruta (absoluta) del directori de treball que s'ha indicat en l'ordre `qsub`.

- PBS_ENVIRONMENT. Indica si el treball és en segon pla (PBS_BATCH) o interactiu (PBS_INTERACTIVE).
- PBS_JOBID. És l'identificador de treball que li assigna el sistema de cues.
- PBS_JOBNAME. És el nom del treball proporcionat per l'usuari.
- PBS_NODEFILE. És el nom del fitxer que conté la llista de nodes que s'assignaran al treball.
- PBS_QUEUE. És el nom de la cua des de la qual s'ha executat el treball.

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

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

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

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

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

L'ordre qstat indica l'estat d'un treball, de les cues i del servidor de cues. L'estat es proporciona mitjançant la sortida estàndard. Quan se sol·licita l'estat d'un treball mitjançant una de les dues opcions que s'han mostrat, qstat retorna la informació de cadascun dels treballs indicats mitjançant el seu identificador o tots els treball d'una destinació donada. Cal tenir present que els treballs en què l'usuari no té privilegis no es mostraran. Quan s'utilitza una de les tres últimes opcions que s'han mostrat, qstat retornarà la informació de cada destinació.

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

L'ordre qdel elimina els treballs especificats en l'ordre en què es proporcionen. Un treball només el pot eliminar o bé el seu propietari o bé l'operador o l'administrador del sistema de cues. Per a eliminar un treball, el servidor enviarà un senyal SIGTERM seguit per un senyal 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...
```

L'ordre qalter modifica els atributs d'un o més treballs a partir del seu identificador. Només es poden modificar les opcions que es mostren en la descripció de l'ordre.

Els treballs s'envien mitjançant *scripts* que permeten comunicar-se amb el programa d'enviament de treballs MPI per a especificar-li la quantitat i quins nodes s'han d'utilitzar dels disponibles per al PBS, segons els requeriments de l'usuari. El *server* del PBS no executarà més treballs als nodes ocupats fins que s'hagi acabat el treball actual. A continuació es mostra un exemple de *script* de PBS per a l'execució d'una aplicació MPI mitjançant `mpiexec` (equivalent a `mpirun`, que hem vist anteriorment).

```
#!/bin/sh
#! Exemple de fitxer de definició de treball per enviar mitjançant qsub
#! Les línies que comencen per #PBS són options de l'ordre qsub

#! Nombre de processos  (8 en aquest cas, 4 per node)
#PBS -l nodes=4:ppn=2

#! Nombre dels fitxers per a la sortida estàndard i error
#! Si no s'especifiquen, per defecte són <job-name>.o<job_number> i <job-name>.e<job_number>
#PBS -e test.err
#PBS -o test.log

#! Adreça electrònica de l'usuari per a quan el treball acabi o s'avorti
#PBS -m ae

#! Directori de treball
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
```

També hi ha altres gestors de cues com ara Loadleveler o SLURM. El Loadleveler és un gestor de cues dissenyat per IBM que es caracteritza per la seva facilitat per a processar treballs en entorns clúster. Permet executar treballs paral·lels (per exemple, MPI) i és fàcilment escalable a milers de processadors. El Loadleveler va ser un dels primers sistemes a incorporar l'algoritme de planificació *backfilling*. SLURM és un gestor de cues de codi obert dissenyat per a clústers Linux de diferents mides.

4.2. Planificació

El problema de la planificació de tasques o treballs en un sistema de computació d'altes prestacions es pot dir que consisteix a, a partir d'una llista de treballs que s'esperen en la cua, les seves característiques –com, per exemple, el nombre de processadors que necessiten, els fitxers executables, els fitxers d'entrada, etc.– i l'estat del sistema, decidir quins treballs s'han d'executar i en quins processadors. Aquest problema es pot dividir en dues etapes que segueixen cadascuna una política concreta: la política de planificació de treballs i la política de selecció de recursos.

Treball

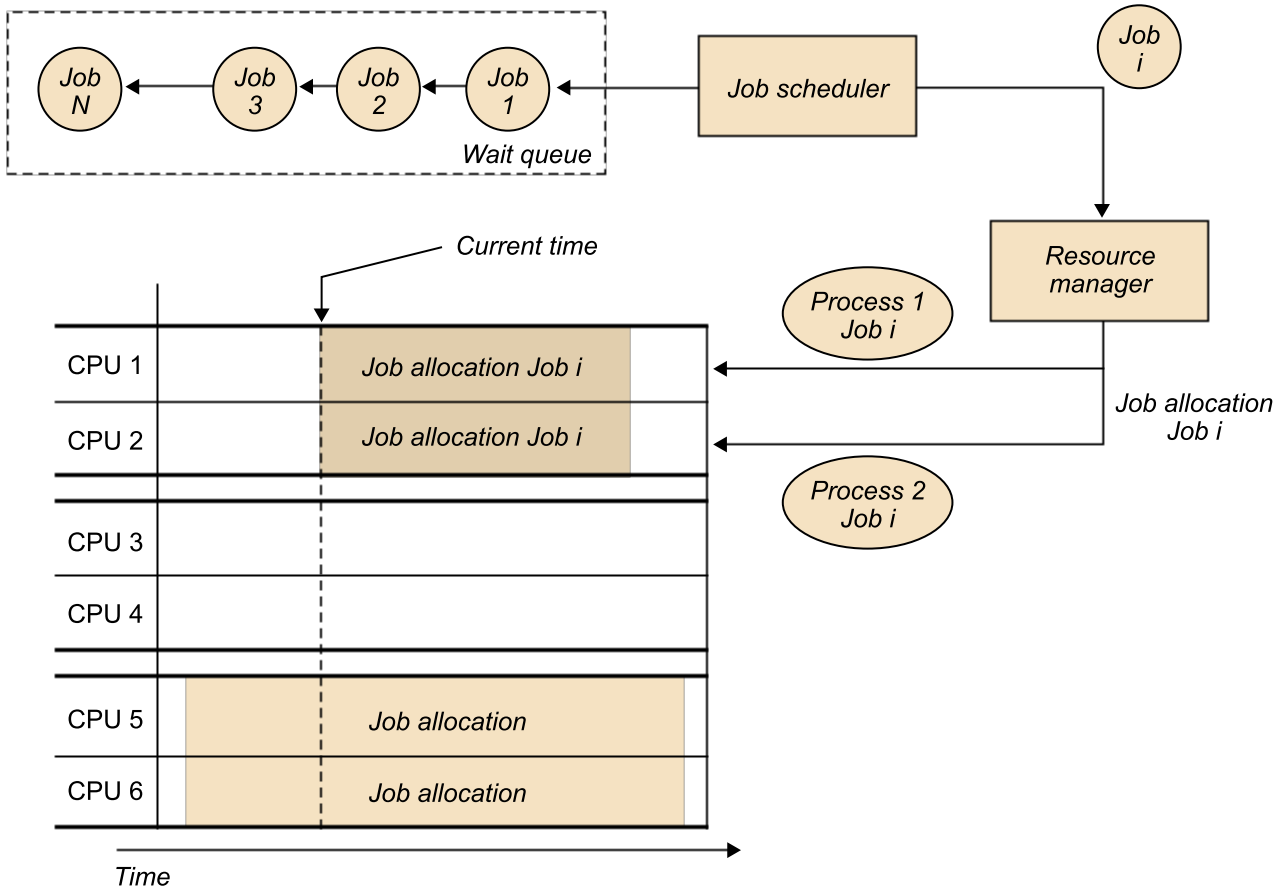
Utilitzarem el terme *treball* per a referir-nos a l'execució d'una aplicació amb uns paràmetres concrets durant tot el subapartat.

Com mostra la figura 21, en la primera etapa de la planificació, el planificador (*scheduler*) ha de decidir quins dels treballs que s'esperen en la cua (*job1*, *job2*, ..., *jobN*) han de començar, tenint en compte el recursos que hi ha disponibles (CPU1, CPU2, CPU3, CPU4). L'algoritme que fa servir el planificador per a fer aquesta selecció s'anomena *política de planificació de treballs*[2]. Un cop s'ha seleccionat el treball més apropiat per a ser executat, el planificador demana al gestor de recursos (*resource manager*) que assigni el treball seleccionat (*jobi*) als processadors més adequats, en funció dels requeriments del treball (en la figura 21, el treball *jobi* requereix dos processadors). Llavors, el gestor de recursos seleccionarà els processadors més apropiats (CPU1 i CPU2 en l'exemple) i assignarà processos del treball a cadascun. El gestor de recursos implementa una política de selecció de recursos[3] concreta.

(2)En anglès, *job scheduling policy*.

(3)En anglès, *resource selection policy*.

Figura 21. El problema de la planificació mitjançant un exemple



S'han proposat moltes tècniques i polítiques de planificació i de selecció de recursos durant les últimes dècades per a optimitzar la utilització dels recursos i reduir els temps de resposta dels treballs. Tal com veurem a continuació, les polítiques de planificació de treballs que més s'han utilitzat en entorns d'altes prestacions són les que estan basades en *backfilling*. S'ha demostrat que aquestes polítiques proporcionen un bon balanç entre el rendiment del sistema (nombre de processadors utilitzats) i el temps de resposta dels treballs. Tot i així, el problema principal de l'algoritme que utilitzen aquestes polítiques és que l'usuari ha de proporcionar una estimació del temps d'execució dels treballs que envia i, normalment, aquestes estimacions no són gens acurades. A més, en moltes situacions l'usuari no té suficients coneixements o informació per a proporcionar aquests paràmetres.

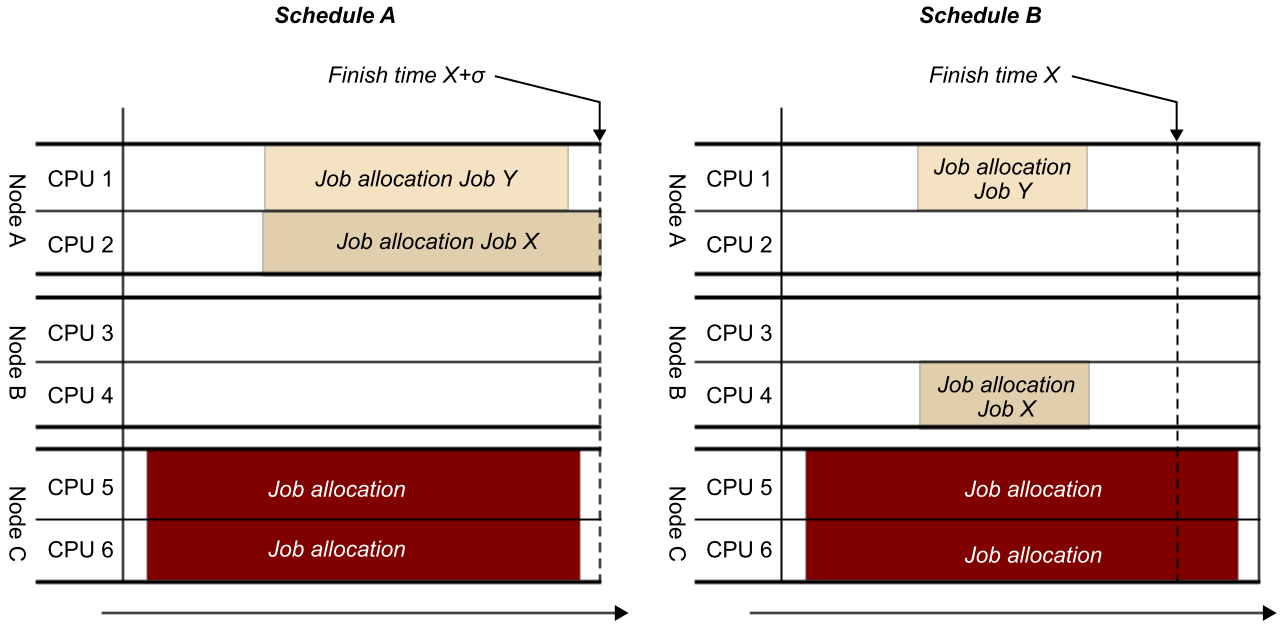
Temps de resposta del treball

És el temps des que s'envia un treball fins que n'acaba l'execució, incloent-hi el temps a la cua.

Per exemple, un usuari no expert en arquitectura de computadors no ha de saber necessàriament que una aplicació pot tenir una certa duració en un tipus d'arquitectura o un 25% més en un altre tipus concret que inicialment pot semblar similar (per exemple, per la quantitat de memòria cau). Tot i així, el temps d'execució d'un treball pot dependre de molts factors com ara el nombre de processadors emprats, l'estat del sistema, les dades d'entrada del programa, etc. També pot succeir que usuaris amb experiència no disposin de suficient informació per a estimar el temps d'execució. Les polítiques de planificació de treballs se centren en la selecció dels treballs que s'han d'executar, però moltes vegades no tenen en compte la ubicació que han de tenir. Aquesta decisió moltes vegades es prenen de manera unilateral mitjançant polítiques de selecció de recursos.

Normalment el que fa el sistema de recursos és ubicar els treballs de dues maneres diferents, o bé un conjunt de filtres predefinits s'apliquen a les característiques dels treballs per a trobar els processadors adequats, o bé els processos dels treballs s'ubiquen en nodes complerts quan l'usuari especifica que els treballs han d'executar-se en nodes no compartits. La primera manera no té en compte l'estat del sistema i, en conseqüència, un treball intensiu de memòria es podria ubicar en un node en què l'amplada de banda a memòria està pràcticament saturada, la qual cosa causaria una degradació important de tots els treballs ubicats en el node. La segona manera normalment implica que la utilització dels recursos és substancialment reduïda perquè normalment els treballs no utilitzen tots els processadors i recursos del node en què estan ubicats. Per tant, en certes arquitectures, quan els treballs comparteixen recursos (per exemple, la memòria) poden resultar amb sobrecàrrega. Això té un impacte negatiu en el rendiment dels treballs que utilitzen els recursos i hi ha un impacte negatiu colateral en el rendiment del sistema. Aquest problema s'il·lustra en la figura 22, en la qual es mostren diferents planificacions dels treballs quan els treballs *job Y* i *job X* s'envien al sistema. Tenint en compte les polítiques de planificació actuals, la planificació més probable seria el cas A. En aquesta, els treballs *job Y* i *job X* s'ubiquen consecutivament en el mateix node (node A). Si els dos treballs són intensius en memòria, hi hauria una reducció en el rendiment de la seva execució deguda a la contenció de recursos. En canvi, si el planificador té en compte l'estat del sistema i els requeriments dels treballs, la planificació més probable seria la del cas B. En aquest darrer cas, tant el treball *X* com el *Y* no tindrien penalització en el rendiment de la seva execució per contenció de recursos.

Figura 22. El problema de la utilització de recursos

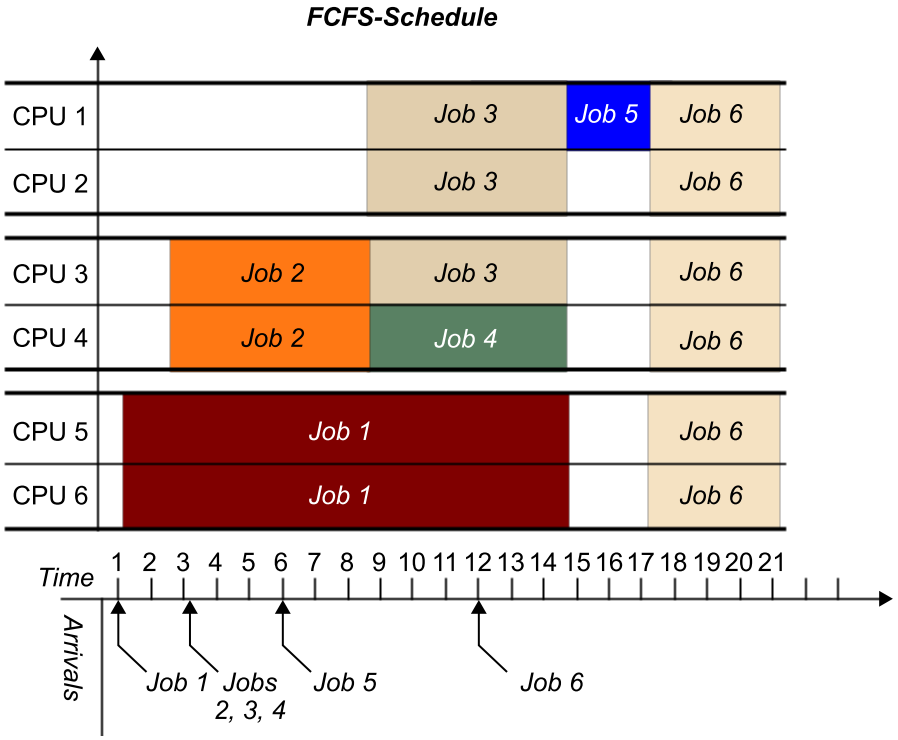


4.2.1. Polítiques de planificació basades en *backfilling*

Des del principi dels anys noranta s'ha investigat molt activament en l'àmbit de la planificació de treballs en computadors d'altes prestacions, i les polítiques basades en *backfilling* són les que s'han utilitzat més extensament en centres d'altes prestacions. El *backfilling* és una optimització de la política de planificació FCFS[4]. En FCFS, el planificador posa en cua tots els treballs enviats en ordre d'arribada i cadascun d'aquests treballs té associat un nombre de processadors requerits. Quan un treball finalitza la seva execució, si hi ha suficients recursos per a començar l'execució del treball següent de la cua, el planificador pren el treball de la cua i el comença a executar. En cas contrari, el planificador ha d'esperar fins que hi hagi prou recursos disponibles i no es pot executar cap altre treball de la cua. La figura 23 mostra una possible planificació de la política FCFS. El problema principal d'utilitzar aquesta política és que el sistema es fragmenta i la utilització dels recursos pot ser molt baixa.

[4]De l'anglès *first-come-first-serve*.

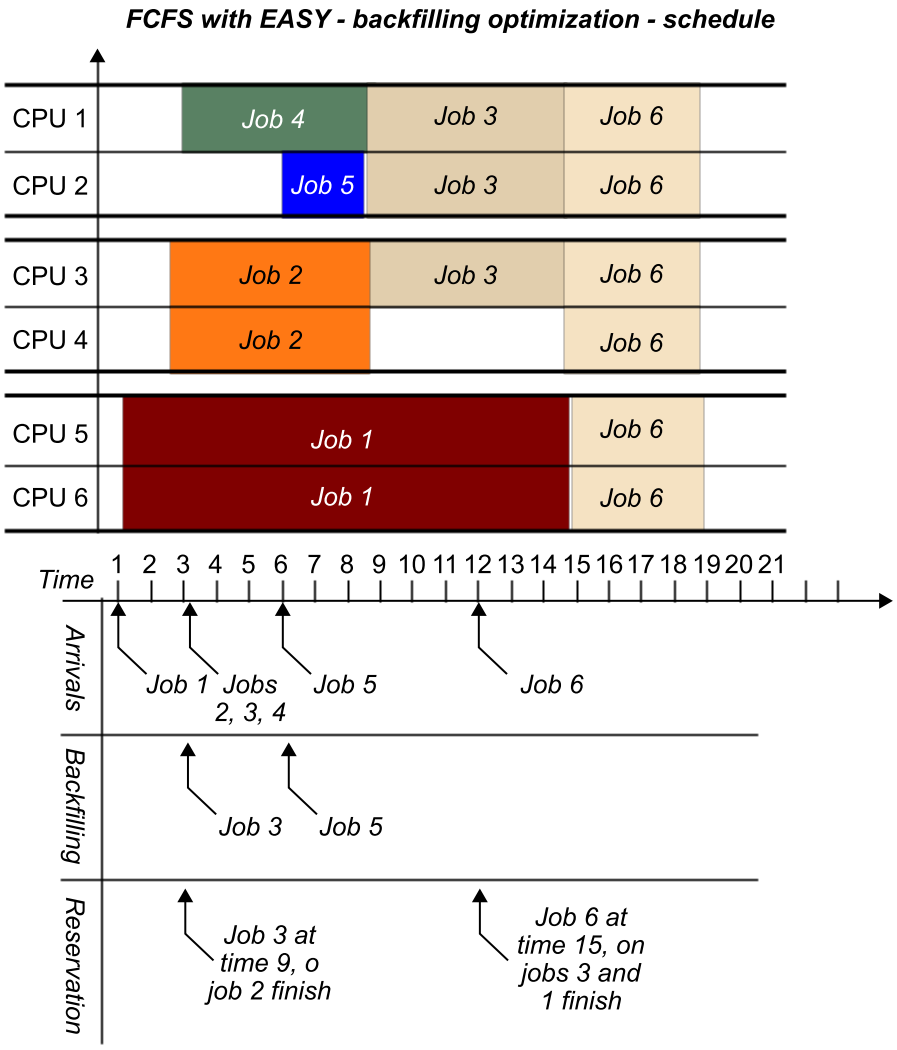
Figura 23. Exemple de planificació basada en la política FCFS



Amb la política de planificació *backfilling* es permet executar treballs que s'han posat a la cua posteriorment a d'altres que ja són en la cua mentre que el temps estimat d'execució dels treballs que ja hi són no s'endarrereixi. Noteu que per a aplicar aquesta optimització, cal saber el temps d'execució aproximat dels treballs quan s'envien (per tant, abans de l'execució), ja que sinó és així no hi ha prou informació per a assegurar que el temps d'execució del treball del principi de la cua no s'endarrereixi. La figura 24 presenta una possible planificació de l'exemple anterior amb optimització mitjançant una política *backfilling*. En aquest exemple, als treballs *job 4* i *job 5* se'ls ha aplicat *backfilling* perquè els treballs *job 3* i *job 6* no poden començar, ja que no hi ha prou processadors

disponibles per a ells. Gràcies a aquesta optimització, la utilització del sistema es pot millorar molt significativament. Molts dels processadors que quedarien aturats sense feina amb una política FCFS poden avançar a partir de *backfilling*.

Figura 24. Exemple de planificació FCFS optimitzada amb EASY *backfilling*

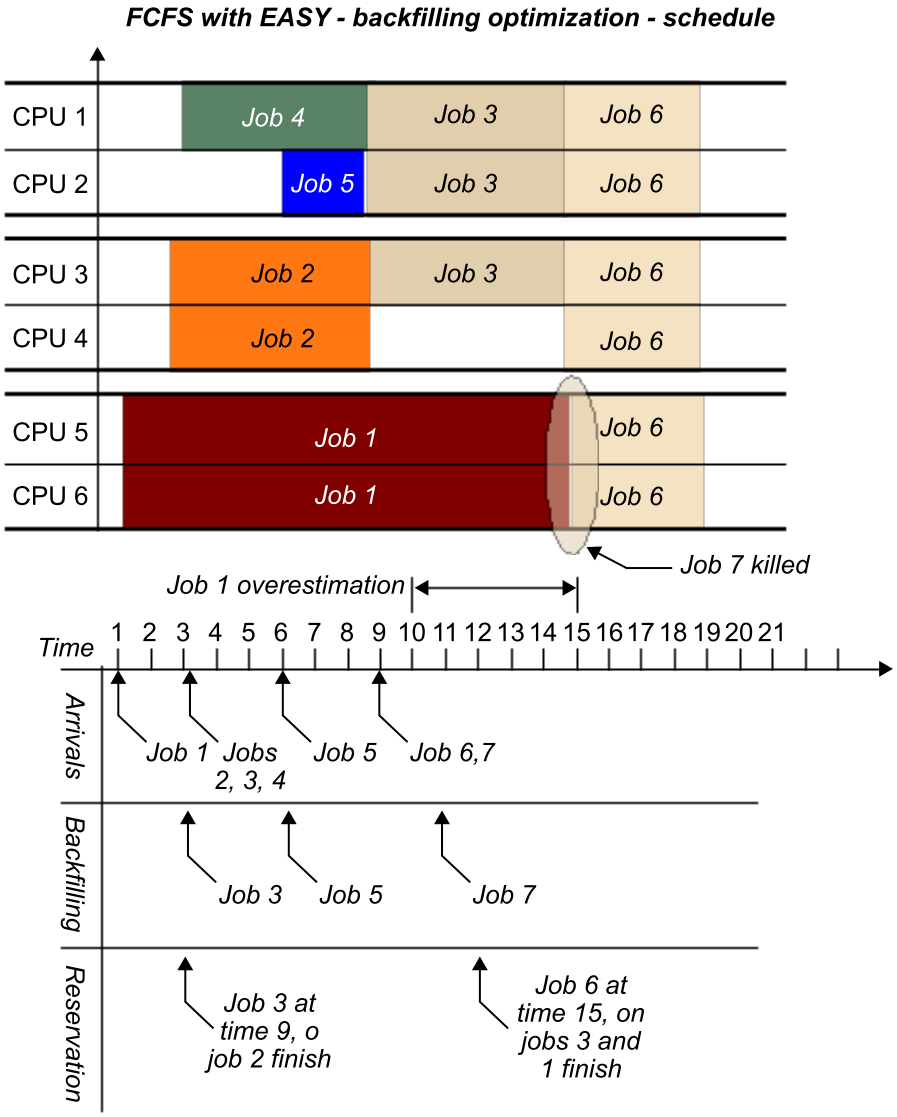


L'inconvenient principal d'aquest algoritme és que l'usuari ha de proporcionar el temps d'execució estimat per al treball quan aquest s'envia al sistema de cues. En situacions en què el temps d'execució que l'usuari ha proporcionat és inferior al temps real d'execució, el treball és matat pel planificador quan detecta que ha excedit el temps d'execució sol·licitat. L'altra situació que es dóna freqüentment en aquest escenari és que el temps d'execució estimat és substancialment superior al real.

En l'exemple de la figura 25 es mostren les dues situacions que s'han descrit anteriorment. En aquesta planificació el treball *job 1* ha estat sobreestimat. Un cop el planificador ha detectat aquesta situació i utilitzant l'estimació de temps d'execució proporcionat per l'usuari, el planificador aplica *backfilling* al

treball *job 7*. Tot i això, el treball *job 7* és matat més endavant, ja que excedeix el temps d'execució especificat. Noteu que en cas que el treball *job 7* no fos matat, el temps d'inici del treball *job 6* podria endarrerir-se.

Figura 25. Exemple de sobreestimació i subestimació de recursos en planificació FCFS optimitzada amb EASY *backfilling*



La política *backfilling* que hem presentat anteriorment és la política basada en *backfilling* més senzilla, que va proposar per Lifka i altres, que s'anomena *EASY-backfilling*. S'han proposat moltes variants d'aquesta política, de les quals podem destacar les que tenen en compte consideracions com, per exemple, l'ordre en què es poden aplicar *backfillings* als treballs. Un exemple és la variant Shortest-Job-Backfilled-First.

Lectura complementària

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, pàg. 41-47).

Activitat

Busqueu informació sobre diferents polítiques i tècniques de planificació de treballs prenent com a referència el treball del doctor Feitelson.

Bibliografia

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* (2a. 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* (1a. 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, pàg. 41-47).