

*Títol:* Adaptació i desenvolupament d'un sistema  
de mesures de rendiment per a MareNostrum

*Volum:* 1

*Alumne:* Xavier Abellán Écija

*Director:* David Vicente Dorca

*Ponent:* Jesús Labarta Mancho

*Departament:* Arquitectura de Computadors

*Data:* 02/12/2008



---

## **DADES DEL PROJECTE**

*Títol del Projecte:* Adaptació i desenvolupament d'un sistema de mesures de rendiment a MareNostrum

*Nom de l'estudiant:* Xavier Abellán Écija

*Titulació:* Enginyeria Informàtica

*Crèdits:* 37,5

*Director/Ponent:* Jesús Labarta Mancho

*Departament:* Arquitectura de Computadors

---

## **MEMBRES DEL TRIBUNAL (nom i signatura)**

*President:* Xavier Martorell Bofill

*Vocal:* Albert Oliveras Lluell

*Secretari:* Jesús Labarta Mancho

---

## **QUALIFICACIÓ**

*Qualificació numèrica:*

*Qualificació descriptiva:*

*Data:*

---



# Adaptació i desenvolupament d'un sistema de mesures de rendiment a MareNostrum

Xavier Abellán Écija

2 de desembre de 2008



## Agraïments

Aquest projecte no hagués estat possible sense l'ajuda i el recolzament de moltes persones. M'agradaria dedicar-lo per tant a totes elles:

A David Vicente, que sempre ha trobat un moment per a discutir i solucionar els dubtes que han anat sorgint i m'ha guiat a través de les etapes més difícils del projecte.

Als meus companys i ex-companys del grup de Suport Jorge Naranjo, Diego Gandía i Pere Munt. Sens dubte han estat una font d'inspiració i d'idees, a més d'oferir-me el seu ajut a l'hora de provar tot el sistema.

Als membres i ex-membres de l'equip de Sistemes Sergi Moré, Jordi Valls, Javier Bartolomé, Juan C. Sanchez, Àlex Font, Miguel Ros i Ernest Artiaga, per la paciència que han demostrat. La seva ajuda ha estat indispensable per dur a terme algunes parts del plantejament inicial, així com tot el desplegament del sistema en les parts finals.

Als desenvolupadors de les eines utilitzades Philip Mucci i Daniel Ahlin. La seva atenció i interès han estat cabdals per a que el projecte hagi funcionat com ho ha fet. A més, les seves aportacions han enriquit i influït d'una forma determinant en el resultat final.

A Roberto Barreda, pel seus consells que han estat tant útils tant per a la implementació com la realització d'aquesta memòria, i per al suport que m'ha brindat.

A la meva família, i a Rosa, per donar-me suport i ànims durant tot aquest temps, a més d'ajudar-me amb la revisió d'aquesta memòria.

I en general, a tothom que m'ha donat un cop de mà per aconseguir que aquesta aventura hagi estat un èxit.

Gràcies a tots.



---

# Índex

<b>1</b>	<b>Introducció</b>	<b>1</b>
1.1	Motivació . . . . .	1
1.2	Objectius . . . . .	2
1.3	Metodologia . . . . .	3
1.4	Organització de la memòria . . . . .	4
<b>2</b>	<b>Entorn del projecte i estat de l'art</b>	<b>7</b>
2.1	El centre i el supercomputador . . . . .	7
2.2	Supercomputació . . . . .	8
2.2.1	Paral·lelisme . . . . .	9
2.2.1.1	Escalabilitat . . . . .	10
2.2.1.2	Models de paral·lelisme . . . . .	11
2.2.2	Xarxes d'interconnexió . . . . .	15
2.2.3	Sistemes de fitxers . . . . .	16
2.2.4	Sistema de cues . . . . .	18
2.3	Mesures de rendiment . . . . .	20
2.3.1	Què es pot mesurar d'una aplicació? . . . . .	20
2.3.2	Mètodes d'obtenció de la informació . . . . .	23
2.4	Aplicacions existents . . . . .	26
2.4.1	IBM HPM Toolkit . . . . .	27

2.4.2	Papiex . . . . .	28
2.4.3	Perfsuite . . . . .	28
2.4.4	MpiP . . . . .	29
2.4.5	Vampir . . . . .	30
2.4.6	Paraver . . . . .	30
2.4.7	Tau . . . . .	31
2.4.8	PerfMiner . . . . .	32
<b>3</b>	<b>Desenvolupament del projecte</b>	<b>35</b>
3.1	Punt de partida . . . . .	35
3.1.1	Requisits generals . . . . .	35
3.1.2	Elecció de l'eina: PerfMiner . . . . .	36
3.2	Estructura general del sistema . . . . .	38
3.3	Extracció de la informació de rendiment . . . . .	40
3.3.1	Anàlisi del funcionament de Papiex . . . . .	40
3.3.2	On escriure la informació obtinguda . . . . .	42
3.3.3	Adaptació . . . . .	45
3.3.4	Instal·lació . . . . .	47
3.4	Integració amb el sistema . . . . .	48
3.4.1	Requisits principals . . . . .	49
3.4.2	Anàlisi i disseny de la solució . . . . .	50
3.4.2.1	Els <i>plugins</i> SLURM: SPANK . . . . .	51
3.4.2.2	Disseny del <i>plugin</i> : spank_perfminer . . . . .	54
3.4.2.3	Nivells d'extracció . . . . .	59
3.4.3	Implementació i instal·lació . . . . .	60
3.5	Recol·lecció de la informació . . . . .	62
3.5.1	Requisits principals . . . . .	62

3.5.2	Anàlisi i disseny de la solució . . . . .	62
3.5.2.1	Estratègia de recollida . . . . .	63
3.5.2.2	Arquitectura de la solució . . . . .	68
3.5.2.3	Disseny de pfmcollector . . . . .	69
3.5.3	Implementació i instal·lació . . . . .	73
3.6	Emmagatzemament a la base de dades . . . . .	75
3.6.1	Requisits principals . . . . .	75
3.6.2	Disseny de la base de dades . . . . .	76
3.6.3	Integració i adaptació . . . . .	79
3.7	Anàlisi i visualització . . . . .	80
3.7.1	Requisits principals . . . . .	80
3.7.2	Disseny de la interfície web . . . . .	81
3.7.3	Adaptació i millora . . . . .	83
<b>4</b>	<b>Experiments i Resultats</b>	<b>87</b>
4.1	Entorns de treball . . . . .	87
4.1.1	Ordinador portàtil estàndard . . . . .	87
4.1.2	Clustertest . . . . .	88
4.1.3	Nord . . . . .	89
4.1.4	MareNostrum . . . . .	90
4.2	Experiments realitzats . . . . .	90
4.2.1	Sobrecàrrega introduïda . . . . .	90
4.2.2	Sistemes de fitxers . . . . .	93
4.2.3	Proves de capacitat . . . . .	95
4.2.4	Proves amb aplicacions habituals . . . . .	99
4.2.4.1	MFLOPS per usuari . . . . .	100
4.2.4.2	MFLOPS per aplicació d'un usuari concret . . . . .	100
4.2.4.3	Anàlisi d'una execució concreta . . . . .	101

<b>5</b>	<b>Planificació i Anàlisi econòmic</b>	<b>103</b>
5.1	Planificació . . . . .	103
5.2	Anàlisi econòmic . . . . .	105
5.2.1	Cost de la implementació . . . . .	105
5.2.2	Cost de l'equipament . . . . .	106
5.2.3	Cost total . . . . .	107
<b>6</b>	<b>Conclusions i Treball Futur</b>	<b>109</b>
6.1	Sumari . . . . .	109
6.2	Treball futur . . . . .	110
<b>A</b>	<b>Descripció de mètriques derivades</b>	<b>113</b>

## Introducció

Aquest projecte vol desenvolupar i adaptar una infraestructura capaç d'extreure informació de rendiment d'aplicacions en entorns de supercomputació que sigui capaç de funcionar en grans clústers com MareNostrum, que es troba al BSC-CNS ( Barcelona Supercomputing Center - Centro Nacional de Supercomputación, <http://www.bsc.es>). A l'hora de dissenyar el sistema s'hauran de tenir en compte certs punts com la integració amb el sistema, la recollida de la informació, el seu emmagatzemament i el posterior anàlisi.

### 1.1 Motivació

Bona part de la investigació o el desenvolupament de nous productes seria impossible sense l'ajut dels ordinadors. Desenvolupament de nous fàrmacs, investigació espacial, disseny de nous avions, anàlisi de models climàtics, etc. són clars exemples de la necessitat de simular mitjançant l'ordinador el món real, ja sigui per motius de viabilitat, o per raons econòmiques. Aquestes simulacions requereixen una gran capacitat de càlcul, de procés i d'emmagatzematge, que fan que sigui necessària la utilització de supercomputadors.

Un supercomputador és, doncs, una eina molt potent al servei de la recerca i el desenvolupament, però també costosa de mantenir, tant en termes de maquinari com de consum energètic. És per això que cal optimitzar al màxim les aplicacions de càlcul que s'executen garantint un ús eficient dels recursos dels que es disposen.

Aquest punt no només és interessant per als que ofereixen els recursos, sinó també pels usuaris que els consumeixen. Habitualment es realitzen assignacions de recursos als usuaris per a que

puguin realitzar la seva tasca. Per realitzar aquestes assignacions, s'utilitzen les hores de càlcul consumides com a unitat de referència. Tenint en compte això, si s'aconsegueix una aplicació més eficient, l'usuari podrà realitzar més càlculs en el mateix temps assignat, augmentant així tant la seva productivitat com la del sistema en general.

Des del punt de vista de l'administrador, és també molt interessant conèixer el comportament de les aplicacions que s'executen ja que es podrà adequar millor el sistema a les necessitats dels usuaris, a l'hora de poder identificar possibles problemes de rendiment en certes aplicacions i per tant intentar millorar-ne l'eficiència.

Per tant, la raó principal per la que s'ha realitzat aquest projecte és la de tenir una eina capaç d'extreure aquesta informació de rendiment a nivell global i poder posteriorment analitzar-la per descobrir els punts on s'ha de millorar i prendre les mesures corresponents.

## 1.2 Objectius

Vista la necessitat de posseir informació de com es comporten les aplicacions que s'executen en un supercomputador, l'objectiu serà arribar a tenir un sistema capaç d'obtenir aquestes dades de les aplicacions d'una manera fàcil i amb el menor impacte amb la configuració original. Per tant, és de vital importància que les aplicacions existents de les que volem obtenir dades no hagin de patir modificacions, ni a nivell de codi, ni a nivell de compilació. Això dificultaria molt la seva integració i el seu ús, tant per a l'usuari com per a l'administrador.

Actualment existeixen eines que permeten extreure informació del rendiment d'aplicacions a diferents nivells de detall, i que seran analitzades en detall a la secció 2.4. Aquestes eines són utilitzades arreu per realitzar anàlisi d'aplicacions, són globalment acceptades i per tant fiables. No es pretén llavors desenvolupar completament una sistema que tindria una funcionalitat que ja ofereixen d'altres molt més complertes i treballades, sinó intentar construir un sistema capaç de satisfer les necessitats esmentades intentant utilitzar peces que ja han estat desenvolupades i provades amb anterioritat.

El principal objectiu és, doncs, partint de la base dels sistemes existents, *analitzar* quin és el que s'adapta millor als requeriments, i posteriorment realitzar una adaptació del mateix per tal que pugui funcionar en l'entorn de producció desitjat. Aquesta *adaptació* anirà des de la instal·lació en l'entorn de producció, la *modificació* de les parts que s'integraran fins el *desenvolupament* complet de les parts que no s'ajustin del tot.

Amb tot, la meta final és construir un sistema que sigui capaç d'obtenir informació del comportament de les aplicacions de manera transparent a l'usuari. D'aquesta manera, s'obtindrà una base de coneixement global de rendiment de tot el programari en un o més supercomputadors i l'ús que fan els diferents usuaris dels recursos assignats. Al mateix temps, l'eina oferirà la possibilitat de detectar certs problemes de rendiment d'una certa aplicació per a poder buscar-hi solucions. És important també, a més de la transparència envers l'usuari, la mínima interferència amb la configuració de la màquina i l'entorn de treball, intentant afegir la mínima sobrecàrrega possible tant en el procés d'extracció com en el de recollida.

## 1.3 Metodologia

La metodologia seguida per realitzar aquest projecte és la següent:

1. Estudi de les eines existents que poden tenir a veure amb el projecte, analitzant així si es poden adaptar a les característiques i requisits del nostre sistema.
2. Un cop triada l'eina, cal fixar-se més en detall en el seu funcionament per saber què es podrà aprofitar, què caldrà adaptar o modificar i què caldrà refer de nou.
3. Es realitzarà un anàlisi dels requisits particulars que haurà de tenir el sistema, i es farà el disseny de l'arquitectura del sistema que pugui satisfer aquests requisits.
4. El sistema quedarà dividit en diverses parts ben diferenciades i que seran tractades independentment com a paquets de treball:
  - (a) Extracció de la informació
  - (b) Integració en el sistema
  - (c) Recollida de la informació
  - (d) Emmagatzematge
  - (e) Presentació de la informació
5. Es començarà a treballar en l'ordre lògic que marquen aquests paquets de treball, ja que cada paquet depèn dels anteriors. En aquest punt caldrà veure quins són els requisits específics d'aquella part, i quins dels requisits generals del sistema s'hauran de tenir en ment al fer el disseny del paquet. Es continuarà realitzant en cada cas l'anàlisi i disseny de la base de la

que es parteix, les modificacions necessàries i si cal, la implementació d'aquelles parts que requereixen ser escrites de nou.

6. Per cada paquet de treball es faran proves individuals de validació per comprovar el correcte funcionament del mateix.
7. Un cop hagin estat validats els paquets per separat, serà necessària la configuració d'un entorn de proves per validar el conjunt del sistema, per més tard realitzar una validació de funcionament del sistema complet.
8. El següent pas és la instal·lació del sistema en un entorn de semi-producció on realitzar proves de capacitat, escalabilitat i càrrega del sistema.
9. Per últim, caldrà posar en producció el sistema desenvolupat i començar a realitzar anàlisi de les dades obtingudes.

## 1.4 Organització de la memòria

Aquest document s'ha dividit en els següents apartats:

1. *Introducció*: en aquesta part es fa un breu repàs als objectius, així com a la motivació que ha portat a la realització d'aquest projecte, la metodologia que s'ha seguit per realitzar-lo i una breu descripció de la organització de la memòria.
2. *Entorn del projecte i estat de l'art*: és on es dona una introducció a tots els conceptes que seran necessaris per poder entendre la resta del document. S'introduiran conceptes de supercomputació, mesures de rendiment i com extreure-les. També quedaran descrites altres eines de mesura o sistemes d'anàlisi de rendiment alternatius.
3. *Desenvolupament del projecte*: aquest capítol descriu tot el treball d'implementació portat a terme, explicant les tecnologies utilitzades, el punt de partida per realitzar el projecte i el procés d'adaptació, modificació.
4. *Experiments i resultats*: un cop processada tota la informació del capítol anterior, aquest apartat recull tots els resultats que se'n deriven, la descripció dels entorns de proves, així com la corresponent validació de l'aplicació. S'explicarà també la metodologia seguida i les eines utilitzades.

5. *Planificació i anàlisi econòmic*: aquest capítol dóna més detall sobre la planificació seguida en la realització del projecte així com el corresponent anàlisi econòmic de la implementació i l'equipament utilitzat.
6. *Conclusions*: aquesta secció sintetitza tots els resultats obtinguts i extreu les conclusions que s'han derivat del treball realitzat. A més planteja quines són les tendències futures que poden tenir els temes que s'han tractat durant el projecte.
7. *Apèndix*: per acabar, es pot trobar una descripció de les mètriques derivades que s'utilitzen en el projecte i com es calculen.



## Entorn del projecte i estat de l'art

Aquest capítol dóna una visió global de com es troba actualment l'entorn en el que es desenvolupa aquest projecte. Primer es fa una breu introducció al centre per al que es desenvolupa el projecte i una descripció del supercomputador MareNostrum.

A continuació, s'exposa tot allò que fa falta per a entendre la resta del projecte. S'expliquen des de conceptes de supercomputació que seran tinguts en compte durant el projecte fins a les mesures que es poden extreure de les aplicacions i com es fa aquest procés.

Per últim, s'estudiaran productes existents actualment que permeten extreure i analitzar el rendiment d'aplicacions, fent una breu descripció del seu funcionament i de les seves principals característiques.

### 2.1 El centre i el supercomputador

El projecte es desenvolupa al **Barcelona Supercomputing Center - Centro Nacional de Supercomputación** (BSC - CNS)<sup>1</sup>. BSC és un consorci públic entre el Ministerio de Educación y Ciencia espanyol, la Generalitat de Catalunya i la Universitat Politècnica de Catalunya (UPC) constituït oficialment a l'abril del 2005, i dirigit pel doctor Mateo Valero, també professor de la UPC. Un dels objectius principals del centre és donar servei a la investigació oferint recursos de supercomputació per al progrés científic.

En aquest sentit, una gran part dels recursos es reparteixen en projectes de gran interès científic, els investigadors dels quals podran gaudir-ne de manera gratuïta. Aquests projectes són seleccionats

---

<sup>1</sup>Més informació a [http://www.bsc.es/plantillaA.php?cat\\_id=1](http://www.bsc.es/plantillaA.php?cat_id=1)

per un comitè independent d'experts en les diferents àrees d'investigació. Aquest comitè d'accés decideix la quantitat de recursos que s'assignen a cada projecte en funció del seu valor científic. La recerca que es duu a terme en camps com l'astronomia, ciències de la terra, física, química o biologia i ciències de la vida. Paral·lelament, des del centre també es potencia la investigació interna en diferents àrees del coneixement com són les ciències de la vida, les ciències de la terra, així com la investigació en computació. Aquesta recerca s'endú una altra part dels recursos disponibles.

El centre alberga i també gestiona el supercomputador **MareNostrum**<sup>2</sup>[MNA], un dels superordinadors més potents del món segons la llista Top500, [http://top500.org/\[top\]](http://top500.org/[top]). Està ubicat a l'antiga capella de Torre Girona al Campus Nord de la UPC, dins d'un habitacle de vidre. Està format per 2560 nodes de computació JS21 i 42 servidors p615, distribuïts en 44 armaris, ocupant una superfície total de 120 m<sup>2</sup>. Cada node de computació, o *blade*, consta de 2 processadors powerPC970MP, cadascun d'ells de doble nucli, amb una freqüència de rellotge de 2.3 GHz, 8 GB de memòria RAM i 36 GB de disc dur local. El sistema operatiu utilitzat a cada node és GNU/Linux en la seva distribució SuSE.

Els servidors de disc proporcionen un total de 280 TB d'espai accessibles des de qualsevol node. Els nodes estan connectats entre si utilitzant diverses xarxes d'interconnexió:

- Xarxa Myrinet<sup>3</sup>: xarxa de gran ample de banda i baixa latència utilitzada per les comunicacions de les aplicacions paral·leles.
- Xarxa Gigabit Ethernet: és la xarxa a través de la qual els sistemes de fitxers remots treballen.
- Xarxa Fast-Ethernet: aquesta xarxa, més lenta, s'utilitza exclusivament per a tasques de manteniment i administració del sistema.

En total, els 10240 processadors de MareNostrum ofereixen un rendiment màxim teòric de 94,21 Teraflops (tot i que el rendiment màxim sostingut és de 63.83 Teraflops executant Linpack) i una memòria principal total de 20 TB.

## 2.2 Supercomputació

Una vegada vist on es desenvoluparà el projecte, és convenient introduir certs aspectes del món de la supercomputació abans d'entrar a descriure el projecte a fons. Com es pot observar, la potència

---

<sup>2</sup>Més informació a [http://www.bsc.es/plantillaA.php?cat\\_id=200](http://www.bsc.es/plantillaA.php?cat_id=200)

<sup>3</sup>Myrinet està desenvolupada per Myricom, <http://www.myri.com/>

que pot oferir un sol ordinador amb un sol processador és molt limitada, i per realitzar certs càlculs s'haurien d'esperar mesos i fins i tot anys abans no fos capaç de donar el resultat esperat.

La solució més senzilla a aquest problema és augmentar els recursos, és a dir, afegir més processadors, més memòria, més disc, etc. De fet, molts dels ordinadors que es poden trobar avui en dia al mercat ja disposen de processadors amb més d'un nucli de procés, o *core*. Aquesta idea aplicada a gran escala dona lloc a la supercomputació tal com la coneixem avui en dia. Un clar exemple d'aquesta filosofia és el mateix MareNostrum, amb 10240 processadors distribuïts en 2560 nodes, tal com s'exposa a la secció 2.1.

A la pràctica, però, no és tan senzill de portar a terme. Teòricament, si es duplica el nombre de processadors, s'hauria també de duplicar el rendiment reduint el temps de càlcul a la meitat, però la realitat és una altra. S'han de tenir en compte les dificultats d'aglutinar tots els nodes que formen el supercomputador per a que treballin com a un de sol, incloent la necessitat de compartir informació entre ells, de sincronitzar-se, de gestionar la totalitat dels recursos, etc.

Per aquest motiu, no tan sols cal fixar-se el maquinari, sinó també amb les aplicacions que hi treballaran a sobre. Els programes de càlcul hauran de tenir en compte com és el sistema on s'executaran, i hauran de ser adaptats en conseqüència.

Tot seguit s'exposen els conceptes més rellevants des del punt de vista d'aquest projecte que tenen a veure amb la supercomputació, com són el paral·lelisme i els sistemes de fitxers. Lògicament, s'haurien de tractar molts més temes per formar-se una idea completa d'aquest món, però s'escapen de l'abast d'aquest projecte. Es parlarà per tant del paral·lelisme, dels sistemes de fitxers que s'utilitzen i de com es gestionen els recursos sense aprofundir en temes com les xarxes d'interconnexió

### 2.2.1 Paral·lelisme

Com s'ha apuntat anteriorment, el secret de la supercomputació radica en que disposem de moltes unitats de procés, però per aprofitar-les cal que totes aquestes unitats tinguin alguna tasca a fer al mateix temps. Així neix el concepte de paral·lelisme. Executar un programa paral·lel no és altra cosa que distribuir la feina entre tots els processadors dels que es disposa per reduir el temps que triga en realitzar el càlcul.

A continuació es detallen alguns conceptes relacionats amb el paral·lelisme, com són l'escalabilitat o els models de paral·lelisme que podem tenir.

### 2.2.1.1 Escalabilitat

L'escalabilitat d'un codi ve determinada per l'evolució dels indicadors de rendiment, per exemple el temps total de l'execució, quan s'augmenten els recursos destinats a aquella execució. L'expectativa ideal seria que el rendiment augmentés de la mateixa manera que augmenten els recursos. En aquest cas ideal, es diu que l'escalabilitat de l'aplicació és lineal. Es poden donar casos en els que el rendiment amb més recursos sigui millor encara que l'ideal, tenint una aplicació que superesca-la, o que té una escalabilitat super lineal. De totes maneres, el més habitual és que el rendiment es vagi degradant conforme augmentem els recursos assignats, fins i tot arribant a extrems en els que l'aplicació es comporta pitjor amb més processadors.

Existeixen diversos motius que porten a aquesta degradació. Per un costat, es poden donar dependències entre les diferents parts del programa que fan que no es puguin executar en paral·lel, i que provoquen una necessitat de comunicació i de sincronització entre elles. D'altra banda, també sol passar que no es pugui repartir de manera uniforme la feina a realitzar (desbalanceig). Degut a tot això, sorgeix una sobrecàrrega o *overhead* degut a les comunicacions i sincronitzacions entre les diferents tasques, a més de la inevitable replicació de codi entre les diferents tasques. És desitjable doncs que els programes a executar siguin el més escalables possible, tant en termes de temps, com d'altres factors com la memòria, el disc, etc.

**Llei D'Amdahl[Amd62]** El guany de rendiment que s'obté és gràcies a que certes tasques d'una aplicació s'executen en paral·lel, reduint així el temps total de càlcul. Tot i això, hi haurà parts que no serà possible executar en paral·lel, i per tant no guanyaran velocitat al augmentar els recursos. Aquesta part no paral·lela és la que acaba establint-se com a factor limitant a l'hora de millorar el rendiment. Aquest concepte és el que s'extreu de la llei d'Amdahl aplicada al paral·lelisme, on  $S$  és el guany o *speed-up* de l'aplicació,  $f$  és la proporció del codi que es paral·lelitzava, i  $P$  és el nombre de processadors utilitzat.

$$S = \frac{1}{1 - f + \frac{f}{P}}$$

D'aquí se'n desprèn també que per poder obtenir guanys significatius al augmentar el nombre de processadors, s'ha de reduir al màxim la part no paral·lela de l'aplicació.

### 2.2.1.2 Models de paral·lelisme

Existeixen diverses solucions *hardware* que donen suport al paral·lelisme, i que comporten també diferents maneres de treballar a nivell *software*. Totes les propostes es basen en augmentar el nombre de recursos (processadors, memòria, disc, etc), però la diferència és com s'interconnectaran entre ells. Fonamentalment, distingim dos models basats en l'arquitectura de memòria que s'utilitza: memòria compartida o memòria distribuïda, breument detallats a continuació.

#### Memòria compartida

En el model de memòria compartida, tots els processadors que formen el sistema estan connectats a la mateixa memòria d'una forma comuna, i per tant tenen la mateixa visió sobre ella. Si un d'ells treballa sobre unes dades de memòria, tots els altres també es veuran afectats per aquest canvi. Aquest fet permet que entre els processadors puguin utilitzar la memòria principal com a mitjà de compartició de dades i comunicació entre ells.

És per aquest motiu que en aquest model podem tenir paral·lelisme a nivell de *thread* o fil d'execució. Un mateix procés pot dividir la seva feina en *threads*, que compartiran els seus mateixos recursos, i s'executaran de manera paral·lela, cadascun a un processador. Com que els *threads* d'un mateix procés comparteixen l'entorn d'execució, la compartició de dades entre ells ve implícita. Tot i això, s'han d'aplicar mecanismes tant a nivell *hardware* com a nivell de programació, que evitin problemes derivats de l'accés concurrent a una mateixa dada, mantenint sempre el comportament correcte i esperat de l'aplicació.

Encara que existeixen moltes implementacions del model de *threads*, com poden ser els *pthread*<sup>4</sup>, la que té més acollida a nivell de supercomputació és OpenMP(<http://openmp.org>). OpenMP[DM98] ofereix una API multiplataforma per construir aplicacions paral·leles sobre memòria compartida en llenguatges de programació C, C++ i Fortran. D'aquesta manera s'aconsegueix un codi portable i escalable a l'hora que s'ofereix una programació de l'aplicació paral·lela força senzilla.

Un exemple d'ús d'aquest model podria ser la paral·lelització d'un bucle senzill d'un programa en C com el que es pot veure a la figura 2.1.

Aquest codi realitza una suma dels vector B i C, i deixa el resultat en el vector A. Executat en un sol processador, aquest hauria d'executar N cops el bucle. Si es disposa de més d'un processador, es podria partir el vector en trossos i repartir-los entre els P processadors disponibles, de manera

---

<sup>4</sup>estàndard de *threads* POSIX

```

for (i=0; i<N; i++) {
    A[i]=B[i]+C[i]
}

```

Figura 2.1: Bucle seqüencial senzill

que cadascun només realitzaria  $N/P$  voltes. OpenMP crearia tants *threads* com se li hagi indicat en la variable d'entorn `OMP_NUM_THREADS`, i cadascun treballaria sobre una part determinada dels vectors. Per aconseguir aquesta meta, només caldria afegir un *pragma* al codi font.

```

#pragma omp parallel for
for (i=0; i<N; i++) {
    A[i]=B[i]+C[i]
}

```

Figura 2.2: Bucle senzill paral·lelitzat amb OpenMP

Com es pot observar a la figura 2.2, el programador no es preocupa de la creació dels *threads* necessaris, ni de la distribució de la feina. Simplement indica el que vol paral·lelitzar, i el compilador s'encarrega d'afegir el codi necessari per a que funcioni. Lògicament, aquests són exemples senzills per demostrar la filosofia de treball de OpenMP, però l'especificació permet fer paral·lelitzacions molt més complexes, distribució de la càrrega entre els diversos *threads*, i altres coses que s'escapen ja de la intenció introductòria d'aquesta explicació.

### Memòria distribuïda

En l'altre extrem trobem el model de memòria distribuïda, on cada processador té assignada una memòria independent de la resta. Aquest fet provoca que, amb aquest sistema, els processadors no tenen cap via de comunicació implícita, ja que un processador no pot accedir a la memòria de la resta. D'aquesta manera, el sistema queda configurat per nodes independents amb el seu processador i la seva memòria, i que s'hauran de connectar a través d'una xarxa d'interconnexió per a poder-se comunicar.

En aquest cas ja no és vàlid el model de *threads*, sinó que caldrà fer servir processos separats, ja que cadascun tindrà la seva memòria assignada totalment disjunta de la dels altres processos. Tot i això, segueix sent necessari un mètode per a poder establir compartició de dades o sincronització si es vol executar una aplicació paral·lela. Caldrà fer-ho a través de la xarxa d'interconnexió, que ha de proporcionar una manera ràpida de transmetre informació d'un punt a un altre. La xarxa haurà de tenir un ample de banda considerable per a poder enviar grans quantitats de dades en un

temps raonable, alhora que ha de garantir una latència el més baixa possible, ja que també s'haurà d'utilitzar com a infraestructura per a la sincronització.

El model de programació més utilitzat actualment, convertit en estàndard *de facto* per a aplicacions paral·leles per a memòria distribuïda és MPI (Message Passing Interface <http://www-unix.mcs.anl.gov/mpi/>). MPI[Mes95] és una interfície de pas de missatges entre diferents processos que proporciona una manera relativament senzilla de paral·lelització d'un programa. Com el seu nom indica, la comunicació entre processos es realitzarà a través d'enviament de missatges, que podran contenir dades per a l'altre procés o sincronització. Actualment existeixen diverses implementacions de les especificacions MPI, com per exemple MPICH[GLDS96] o OpenMPI[GWS05], entre d'altres.

Qualsevol aplicació MPI funcionarà amb independència de la màquina on s'executa, aportant així un grau important de portabilitat. A més, també proporciona eficiència i escalabilitat, ja que la implementació de les rutines de la llibreria MPI són optimitzades per a cada arquitectura on s'executa. Els llenguatges de programació que disposen d'implementació d'aquesta llibreria són, com en el cas de OpenMP, C/C++ i Fortran, ja que són els llenguatges de programació més utilitzats en entorns de supercomputació i càlcul científic. Tot i això, també hi ha implementacions parcials de l'estàndard en llenguatges com Python, Ocaml o Java.

La llibreria disposa de rutines per obrir el paral·lelisme i tancar-lo, així com funcions d'enviament i recepció de missatges de diversos tipus i mètodes de sincronització. Amb tot, s'ofereix al programador una gran varietat de possibilitats que responen als casos més habituals de comunicació i transferència d'informació que es duren a terme d'una manera molt eficient.

La paral·lelització del codi és més complexa que amb OpenMP, ja que ara el programador sí que ha de tenir en compte com es realitza la compartició de les dades, indicant explícitament el que s'envia o es rep. També cal vetllar per la sincronització entre els diferents processos per evitar possibles problemes derivats de l'ordre com s'executen els diferents processos. Tot i semblar un desavantatge, aquesta filosofia permet fer una paral·lelització a mida de cada aplicació, ja que s'adapta molt bé a qualsevol particularitat del codi.

A la figura 2.3 es pot observar l'estructura bàsica d'un programa MPI en llenguatge C. En concret, es formarà un anell entre tots els processos que formen l'execució paral·lela, en el que cadascun rebrà una dada de l'anterior i l'enviarà al següent fins a completar una volta sencera. Cal notar que el programa és independent de amb quants processos s'executarà.

Per executar aquest tipus de programes, cal utilitzar la infraestructura que es proporciona en cada instal·lació de l'entorn MPI. Habitualment existeix una comanda com "mpirun", mitjançant la

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
    MPI_Status status;
    int rank, size, tag, next, from;
    tag = 201;

    // Obrim el paral·lelisme
MPI_Init (&argc, &argv) ;

    // Quin procés sóc (rank) i quants processos hi han (size)
MPI_Comm_rank (MPI_COMM_WORLD, &rank) ;
MPI_Comm_size (MPI_COMM_WORLD, &size) ;

    // Calculem a qui hem d'enviar (next)
    // i de qui hem de rebre (from)
    next = (rank + 1) % size;
    from = (rank + size - 1) % size;

    // El procés 0 començarà enviant al procés
    // següent en l'anell (el 1) un enter
    if (rank == 0)
        MPI_Send (&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD) ;

    // Esperem la rebuda d'un enter pel procés anterior a l'anell
MPI_Recv (&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status) ;

    // Després de rebre de l'anterior, cada procés
    // envia al següent en l'anell
MPI_Send (&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD) ;

    // El procés 0 haurà d'esperar l'últim enviament
    // de l'últim procés abans d'acabar
    if (rank == 0)
        MPI_Recv (&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status) ;

    // Tanquem el paral·lelisme
MPI_Finalize () ;

    return 0;
}
```

Figura 2.3: Programa mpi que es comunica seguint la forma d'un anell

qual podem arrencar l'aplicació usant un determinat nombre de CPUs als nodes on es desitgi. També és molt freqüent que s'utilitzin sistemes de gestió dels recursos<sup>5</sup> per a l'execució d'aquestes aplicacions, on és possible que calgui emprar una eina pròpia del gestor de recursos per a treballar amb aplicacions paral·leles MPI.

### Model Híbrid

Al bell mig dels models de memòria compartida i distribuïda purs, trobem el model híbrid. És la combinació dels dos models explicats, en el que la màquina en qüestió estaria formada per un cert nombre de nodes amb memòria independent (memòria distribuïda), a la vegada que a cada node hi tenim diversos processadors que comparteixen els recursos (memòria compartida).

No és gens estrany que aquest model sigui el majoritari quan parlem de supercomputadors, ja que el cost de màquines de memòria compartida amb molt processadors és molt gran, degut a la complexitat d'aquesta compartició entre tants elements. La complexitat en el disseny implica també pèrdua de rendiment com més processadors, disminuint així l'escalabilitat. Tot i això, amb pocs processadors és molt més eficient i ràpida l'execució en memòria compartida (no s'ha d'utilitzar una xarxa per comunicar-se amb un altre node), pel que el disseny mixt és una bona solució. Un clar exemple el trobem al supercomputador MareNostrum<sup>6</sup>.

En el camp de les aplicacions paral·leles, és habitual també que les implementacions de les llibreries MPI continguin optimitzacions per aprofitar el fet de tenir memòria compartida a cada node, pel que els processos que s'executin en un mateix node puguin comunicar-se utilitzant la mateixa memòria, molt més ràpida que la xarxa. Per altra banda, també existeixen aplicacions híbrides, on el paral·lelisme s'obre tant a nivell de procés com a nivell de *thread*. Això vol dir que, per exemple, en una màquina amb 4 processadors per node, es podria arribar a executar una aplicació que col·loqués un procés a cada node, i aquest s'obris en 4 *threads*, gaudint així dels avantatges que ofereixen. D'aquest tipus podem trobar combinacions de MPI amb qualsevol model de *threads*, com OpenMP.

## 2.2.2 Xarxes d'interconnexió

En el cas de supercomputadors com MareNostrum, cal interconnectar tots els nodes que el formen de manera que les comunicacions descrites a la secció 2.2.1.2 es puguin produir de la manera més eficient possible. Concretament, cal donar suport tant a les comunicacions entre nodes com al

---

<sup>5</sup>vegeu la secció 2.2.4

<sup>6</sup>vegeu la secció 2.1

sistema de fitxers compartit<sup>7</sup>. Cadascuna té unes necessitats diferents per a la xarxa. Si bé una xarxa convencional Gigabit Ethernet ja fa el fet per al sistema de fitxers, quan es tracta de les comunicacions MPI no n'hi ha prou.

Tal com es descriu a la secció 2.1, MareNostrum consta de dos tipus de xarxes que interconnecten els nodes del supercomputador. La xarxa Gigabit Ethernet servirà la imatge del sistema operatiu i el sistema de fitxers paral·lel, mentre que la xarxa Myrinet donarà suport a les comunicacions MPI. Myrinet és una xarxa d'alta velocitat (actualment hi ha instal·lada la versió de 2 Gbps), i de baixa latència, produïda per la companyia Myricom[Myr].

L'arquitectura d'aquesta xarxa permet la seva escalabilitat fins a connectar tots els nodes del *clúster*, oferint alhora camins alternatius per si algun enllaç falla. La major part del procés es fa, a diferència de la xarxa Ethernet, a la mateixa targeta de xarxa, que realitza l'accés a memòria utilitzant DMA. El protocol utilitzat també canvia, ja que tot i que Myrinet suporta enviar tràfic convencional TCP/IP, per aprofitar al màxim les seves capacitats s'ha d'utilitzar un protocol especial dissenyat específicament per aquest tipus de xarxes.

Els protocols que actualment es poden utilitzar estan dissenyats per la mateixa companyia però són d'especificacions públiques. Es coneixen com GM i MX, essent aquest últim el més nou i el que dona un millor rendiment. Aquests protocols condensen els nivells 2, 3 i 4 del model OSI[DZ83], proporcionant el control de l'enllaç, l'encaminament i el control de flux i d'errors. De fet, hi ha implementacions de llibreries de *sockets* sobre GM i MX, semblants als habituals *sockets* TCP.

Existeixen també implementacions de l'especificació MPI de MPICH, optimitzades per a cadascun dels protocols. D'aquesta manera, les aplicacions que s'executen al supercomputador utilitzen aquestes implementacions: MPICH-GM i MPICH-MX. Cal remarcar que la xarxa Myrinet només s'utilitza per aquestes comunicacions MPI, evitant qualsevol interferència aliena a les comunicacions de les pròpies aplicacions MPI que pugui afectar negativament al seu rendiment.

### 2.2.3 Sistemes de fitxers

A més del fet que els diferents processos d'una aplicació paral·lela es puguin comunicar i sincronitzar, també és desitjable que tinguin la visió d'un sistema de fitxers comú, de forma que puguin treballar sobre els mateixos arxius tant d'entrada com de sortida. Una primera aproximació seria utilitzar un sistema de fitxers compartit i muntat a través de la xarxa com NFS o Samba, però el rendiment que oferirien seria molt pobre i més si es té en compte la mida dels clústers on es

---

<sup>7</sup>vegeu secció 2.2.3

vol treballar. Aquesta solució no escala a supercomputadors on hi ha milers de nodes que estaran contínuament realitzant operacions d'entrada / sortida, ja que quedaria totalment superat pel gran volum de transaccions a realitzar.

Afortunadament, existeixen sistemes de fitxers pensats per ser utilitzats en entorns paral·lels com supercomputadors, que tenen en compte tant l'escalabilitat com l'eficiència en aquestes circumstàncies. Existeixen diversos productes al mercat d'aquestes característiques. Un exemple és Lustre[Mic07], desenvolupat i mantingut per Sun Microsystems. La seva llicència és GPL i permet oferir un sistema de fitxers paral·lel a desenes de milers de nodes.

Tot i això, no es tractaran aquests altres sistemes, ja que la intenció d'aquesta secció és introduir al lector als conceptes o tecnologies que s'utilitzaran al llarg del projecte. Per aquest motiu, l'estudi es centrarà en el sistema propietari de IBM, el *Global Parallel FileSystem* (GPFS)[F S02], que és el sistema de fitxers utilitzat als clústers del centre com MareNostrum.

GPFS permet muntar un mateix sistema de fitxers en tot un conjunt de nodes d'un clúster, i permet que tant els fitxers com les metadades del sistema de fitxers puguin ser accedides en paral·lel, tot conservant en gran mesura el comportament d'un sistema de fitxers POSIX. D'aquesta manera, s'aconsegueix l'escalabilitat necessària en el rendiment en grans clústers.

A grans trets, el secret de la gran escalabilitat del sistema resideix en la seva arquitectura de discos compartits, que seran accessibles per tots els nodes de computació. Els fitxers es guardaran en trossos repartits entre diversos discs, de manera que es permet l'accés en paral·lel tant de lectura com d'escriptura al fitxer per part d'un gran nombre de nodes. Això requereix una gran sincronització per tal que no es produeixin corrupcions ni en les dades ni en les metadades.

Per dur a terme la sincronització, GPFS proporciona un sistema de bloquejos, o *locks*, distribuït, que permet gràcies a les característiques de l'arquitectura de discos compartits fins i tot que dos nodes escriguin en parts diferents d'un mateix fitxer a la vegada. Tot i això, s'observen alguns problemes de rendiment quan s'intenten escriure molts fitxers en paral·lel sota el mateix directori. El motiu és senzill: tots els nodes han d'adquirir el bloqueig sobre el directori per a poder crear-hi un fitxer al mateix temps, provocant contenció en l'escriptura. La solució clàssica a aquest problema és establir una estructura jeràrquica de directoris, de manera que a cada subdirectori hi escriuran menys processos i per tant augmentant el rendiment, ja que es redueix la contenció.

A banda dels sistemes de fitxers compartits, com GPFS, els nodes que formen el supercomputador poden tenir un disc dur local amb un sistema de fitxers també local i no compartit amb la resta de nodes. Aquest és el cas de MareNostrum, on cada node té un espai local de disc anomenat *scratch*,

amb 30 GB de capacitat. Aquest sistema de fitxers s'utilitza com a lloc temporal on guardar dades mentre dura l'execució d'una aplicació.

Tot i això, de vegades és interessant poder utilitzar *scratch* per tal de no sobrecarregar el GPFS com en el cas d'escriure milers de fitxers dins un mateix directori. En aquests casos, però, s'ha de tenir en compte com copiar les dades cap a cada disc dur de cada node o com recuperar-les de manera que no es penalitzi el rendiment o l'estabilitat del sistema. A més, una altra diferència que pot resultar notable quant al rendiment és la mida del bloc. Mentre que a GPFS, aquesta mida és de 1 MB (enfocat a fitxers grans), a *scratch* el bloc és de 4 KB, molt més eficient per a tractar fitxers petits.

### 2.2.4 Sistema de cues

Els supercomputadors actuals ofereixen una gran capacitat de càlcul, i estan al servei de la recerca i el desenvolupament que duen a terme molts investigadors. Tenint en compte que es disposa d'una gran quantitat de recursos, i que hi haurà molts usuaris que hauran de compartir-los, seria impensable que cadascú que volgués utilitzar el supercomputador hi accedís i utilitzes els recursos (tant de temps com de processadors o nodes) que volgués sense cap tipus d'ordre ni concert. Per tant, esdevé imprescindible comptar amb alguna eina que permeti acomodar tota la demanda als recursos disponibles en cada moment de la manera més eficient possible. Això és el que es coneix habitualment com un sistema de cues.

Des del punt de vista de l'usuari, els seus càlculs no poden ser executats de qualsevol manera, sinó que ha d'enviar el que es coneix com a treball (o *job*, en anglès) al sistema de cues. Un treball és la unitat d'assignació de recursos per excel·lència del sistema de cues, i conté l'execució d'un o més càlculs d'un usuari i que utilitzaran uns mateixos recursos. A la pràctica, el treball s'implementa com un simple *script*, per exemple de bash o tcsh, amb unes directives especials que són les que interpretarà el sistema de cues per assignar els recursos, i que contindran dades com la quantitat de processadors sol·licitada, o el temps mínim durant el qual han de ser reservats. La sortida del programa no apareixerà per pantalla, sinó que quedarà escrita en uns fitxers de sortida.

Un cop encuat el treball, el sistema decidirà on i quan s'executarà, tenint en compte el que ha demanat, els altres treballs que també han estat encuats i els recursos disponibles. Realitzar aquesta gestió no és una tasca senzilla, i existeixen moltes polítiques de planificació de treballs, ja sigui per establir prioritats per a certs usuaris, definir diferents cues on enviar el treball, omplir petits forats tant espacials com temporals (*backfilling*) o garantir una compartició justa dels recursos (*fairsharing*). Aquestes polítiques són un tema interessant i alhora important per al funcionament

del sistema, però no són especialment rellevants per la comprensió de la resta del projecte i no seran tractades en aquesta memòria.

En l'actualitat, la funció del sistema de cues a MareNostrum és compartida per dos aplicacions diferents, però complementaries: Slurm i Moab. Slurm[YJG03] actua com a gestor de recursos, i és per tant qui decideix on col·locarà els treballs que li arriben en funció del que hagi demanat. És l'encarregat també començar l'execució dels diferents aplicacions que componen el treball, així com oferir l'entorn necessari per a que les aplicacions paral·leles es puguin executar en tots els processadors que té assignats. Slurm és programari lliure sota llicència GPL, i té com a principals desenvolupadors i mantenidors al Lawrence Livermore National Laboratory (LLNL) i HP.

Per altra banda, Moab[Res] s'encarrega de la planificació temporal en base a les dades que li proporciona Slurm. És qui decideix quan s'executa un treball tenint en compte diversos criteris de planificació, reserves de nodes, prioritats dels usuaris, etc. Moab és un producte de l'empresa Cluster Resources que requereix de llicència de pagament. La mateixa empresa també proporciona Maui, un planificador de codi obert més senzill que Moab i que no té tantes funcionalitats.

Un exemple de treball que s'envia a les cues de MareNostrum el trobem a la figura 2.4. Es pot apreciar que dins un *script* de bash, hi trobem les directives en forma de comentaris seguits d'una @. Apareixen el nom del job, el directori inicial que es desitja pel job, on deixar les sortides del programa (tant l'estàndard com la d'error), quants processadors es volen demanar i per quant de temps. En concret, aquest treball demanaria 8 processadors durant 2 minuts. A més, per tal d'executar correctament l'aplicació MPI, és necessari arrencar-la amb la comanda "srun" davant. Aquesta utilitat és part del gestor de recursos Slurm, que és la que permet l'execució paral·lela de les 8 tasques que componen l'execució de l'aplicació MPI. En cas de no utilitzar "srun", l'aplicació només s'executaria al primer dels processadors assignats, provocant un error en cas que sigui una aplicació MPI (ja que no té configurat l'entorn adient).

```
#!/bin/bash
# @ job_name           = test_parallel
# @ initialdir         = .
# @ output             = mpi_%j.out
# @ error              = mpi_%j.err
# @ total_tasks        = 8
# @ wall_clock_limit   = 00:02:00

srun aplicacio-mpi parametres-entrada
```

Figura 2.4: Exemple de treball paral·lel a MareNostrum

La sintaxi utilitzada per als treballs no és proporcionada ni per Slurm ni per Moab, sinó que és un mètode desenvolupat al centre per facilitar la feina a l'usuari final, ja que la interfície que ofereix Slurm és més complexa. La comanda per encuar un treball és "mnsbmit", que tradueix la sintaxi vista del treball a la comanda corresponent de Slurm.

## 2.3 Mesures de rendiment

Un dels motius pels quals s'ha desenvolupat aquest projecte és la necessitat de conèixer com es comporta una aplicació en un determinat entorn d'execució. Es fa necessari doncs comptar amb algun procediment per avaluar-la i quantificar el rendiment obtingut dels diversos recursos que ha utilitzat.

En aquesta secció es farà una breu introducció a les mesures de rendiment del programari, tot explicant quina informació útil es pot extreure de l'execució d'una determinada aplicació i com es pot obtenir, a més de posar de relleu les conseqüències de realitzar aquestes mesures.

### 2.3.1 Què es pot mesurar d'una aplicació?

Pràcticament tota característica o peculiaritat d'un programa pot ser quantificada d'alguna manera, si es troba la mètrica adient. Una mètrica no és més que la mesura en unes determinades unitats d'una d'aquestes característiques intrínseques de l'aplicació i la seva execució. De mètriques n'hi ha de molts tipus i per mesurar diferents propietats, ja que la varietat d'usos i camps en els que s'utilitzen programes informàtics fa que cada aplicació es regeixi per uns paràmetres diferents.

Així, no s'utilitzaran les mateixes mètriques per avaluar el rendiment en un joc d'ordinador 3D que en un programa de transferència de fitxers a través de la xarxa. En el primer, hi intervindran conceptes com els *frames* (imatges) per segon que és capaç, mentre que en el segon seria més interessant els bytes per segon que es transmeten. Com es pot observar, el rendiment depèn fortament també de l'entorn on s'executa el programa, ja que, per exemple, el comportament del programa de transferències pot no ser el mateix si es canvia per exemple el tipus de xarxa on es connecta.

Fins i tot es pot parlar de mètriques relacionades amb el cost econòmic derivat de l'execució de l'aplicació, o el seu consum energètic. Tanmateix, en aquest document l'estudi es centrarà en les mètriques més interessants aplicables a programes de càlcul com els que s'executen en entorns de supercomputació. Cal notar que no figuraran ni molt menys totes les mesures que es poden

arribar a obtenir, sinó més aviat un resum d'aquelles mètriques més habituals i útils, ja que també és freqüent construir mètriques derivades a mida a partir d'altres mètriques.

En l'àmbit de la supercomputació, on la gran majoria d'investigadors que hi treballen tenen un cert nombre d'hores de càlcul assignades, una de les mètriques més importants és el **temps total** d'execució del programa. A l'usuari li interessa reduir al màxim aquest temps ( $T_{total}$ ) tot realitzant els mateixos càlculs, de manera que consumeixi els recursos d'una manera el més eficient possible. Aquest temps pot venir donat tant en les unitats clàssiques (hores, minuts, segons, microsegons, etc) com en cicles de rellotge del processador. Passar d'una unitat a l'altre és senzill, utilitzant la freqüència de rellotge del processador ( $f$ ):

$$T_{total} = \frac{\#cicles}{f}$$

Dins del temps, es pot distingir el **temps real** (el que ha passat des que el programa ha començat fins que ha acabat), o el **temps de procés** (el temps que el procés ha estat utilitzant el processador). Cal notar que aquests dos temps no tenen perquè ser iguals, ja sigui per culpa de la mateixa aplicació (realitza molta entrada / sortida i bloqueja el procés), o per culpa de l'entorn (la planificació del sistema operatiu treu el procés de la CPU per posar-ne un altre. Per tal de garantir l'ús òptim dels recursos, és necessari que el temps de procés s'acosti al màxim al real, evitant situacions com les descrites.

A més del temps d'execució, una mètrica també molt important a l'hora de mesurar el rendiment és el nombre d'instruccions<sup>8</sup> que s'han realitzat. Les instruccions realitzades per si soles poden semblar poc interessants, però se sol combinar aquesta dada amb el temps d'execució o els cicles consumits, donant com a resultat una mètrica com **IPC** (Instruccions per cicle), o la seva inversa, **CPI** (cicles per instrucció). Un equivalent a IPC són els milions d'instruccions per segon, o **MIPS**, que mesura exactament el mateix però canviant els cicles per la seva equivalència en segons. Com que els processadors actuals són capaços de fer moltes operacions cada segon, s'adopta el milió d'instruccions com a base per a aquesta mètrica.

$$IPC = \frac{\#instruccions}{\#cicles} \quad CPI = \frac{\#cicles}{\#instruccions} \quad MIPS = \frac{\#instruccions \times 10^6}{T}$$

Els processadors d'avui en dia ofereixen la possibilitat teòrica de realitzar més d'una instrucció per cicle, i per tant el rendiment de l'aplicació serà millor si és capaç d'executar més instruccions

<sup>8</sup>Quan es parla d'instruccions es fa referència a cada una de les operacions que realitza el processador, i no cada línia de codi d'un programa.

en el mateix temps (major IPC), acostant-se tant com sigui possible al limit ideal que ofereix el processador.

Existeix també un subconjunt d'operacions dins les instruccions comuns del processador que són les que realitzen càlculs amb nombres reals (anomenats en la informàtica nombres en punt flotant, degut al mètode que s'utilitza per poder guardar-los i treballar amb ells). Se solen anomenar operacions en punt flotant o ***Floating Point Operations***. Aquesta dada és important des del punt de vista del tipus d'aplicacions que es tractaran, ja que són majoritàriament de càlcul científic, on bona part del temps es dedica a realitzar operacions sobre nombres reals.

Anàlogament a la mètrica MIPS, dividint el nombre d'operacions en coma flotant amb el temps total d'execució s'obté una de les mètriques més conegudes en el món de la supercomputació: les operacions en punt flotant per segon, o **FLOPS**. Com que en l'actualitat s'obtenen rendiments de moltes operacions d'aquest estil cada segon, és habitual veure les unitats múltiples del FLOP en el sistema internacional de mesures, com MegaFLOPS, GigaFLOPS, TeraFLOPS, etc. No cal dir que cal maximitzar aquesta mètrica per obtenir un comportament ideal de l'aplicació mesurada. Habitualment s'utilitza aquesta mètrica per mesurar les aplicacions d'alt rendiment, així com a mesura per a poder classificar els supercomputadors segons la seva potència de càlcul, com és el cas del *ranking* mundial top500, <http://www.top500.org>.

Seguint amb l'ús que fa el programa del processador, hi hauria infinitat de mètriques que serien molt útils per detectar certs problemes de rendiment, i que requereixen entrar més en detall al mateix processador. Un exemple són les mètriques que mesuren el comportament d'aquest component executant un cert codi, com les **fallades de la memòria cau** (tant de nivell 1 com de nivell 2) o *cache misses*. Per a que un codi vagi ràpid és indispensable fer un bon ús d'aquesta memòria, minimitzant el nombre de fallades i evitant així que es perdi molt temps en buscar dades a memòria principal. A més de la memòria cau, també es poden mesurar les **fallades de la TLB**, que també poden conduir a una degradació del rendiment si es produeixen freqüentment.

En la mateixa línia hi hauria mètriques per mesurar el comportament del predictor de salts i el *pipeline* del processador, com el **nombre de salts mal predits** (*missprediction branch*), o el **nombre d'instruccions avortades** quan encara no havien acabat. També podem trobar aquí del nombre d'instruccions de cada tipus (**instruccions d'enters**, *floating point* o *load/store*).

Les aplicacions utilitzen més recursos a part del processador, com la memòria, el disc, o la xarxa. D'aquests altres components també se'n poden extreure dades importants per caracteritzar un cert programa. És interessant conèixer la **quantitat de memòria** que consumeix el procés, ja que un procés que utilitzi molta memòria pot requerir d'un tractament especial alhora d'acomodar-lo en

un cert entorn. La memòria consumida pot ser **memòria resident**, **memòria virtual**, espais de **memòria compartida** (*shared*), espai de **pila** (*stack*), etc. Quant al disc, seria interessant conèixer si l'aplicació en fa un ús intensiu sabent la quantitat d'informació escrita o llegida al disc, o el temps que dedica a fer entrada / sortida, ja sigui en nombres absoluts o relatius a la duració total del procés en qüestió.

A part de la utilització de tots els recursos, quan l'aplicació és paral·lela pot ser interessant conèixer el rendiment de la part MPI. Cal tenir en compte que molts cops el rendiment d'aquestes aplicacions no és el desitjat degut a llargues esperes per sincronització del programa, o la forma com es produeixen les comunicacions. D'aquí se'n poden extreure moltes dades, com el **temps gastat acumulat de cada crida MPI** (enviaments, recepcions, esperes, col·lectives, etc), ja sigui en termes relatius en funció del temps total o absoluts.

El **patró de comunicacions** és un punt interessant de conèixer sobre aquest tipus d'aplicacions, ja que pot determinar en gran mesura el comportament del programa. El patró de comunicacions és la forma com cada tasca de les que componen l'execució MPI es comunica amb les altres, la quantitat d'informació que envia i rep de cadascun dels altres processos.

### 2.3.2 Mètodes d'obtenció de la informació

Està clar que per poder estudiar i avaluar les mètriques descrites cal poder-les mesurar d'alguna manera. Tant el maquinari com el sistema operatiu ofereixen en el seu entorn eines per a poder consultar aquest tipus d'informació. Tot seguit es descriuran breument una sèrie de mètodes que permeten la obtenció d'aquestes dades, així com posar de manifest les conseqüències associades que té realitzar les mesures.

Com ja s'ha apuntat, el mateix sistema operatiu, en conjunt amb el suport *hardware* necessari, és capaç d'oferir un rellotge que permet conèixer l'instant exacte de temps (normalment amb precisió de microsegon) en un moment determinat. D'aquesta manera, es pot obtenir el temps total d'execució d'un cert programa. Linux ofereix per aquest propòsit les crides *gettimeofday* i *getrusage*. Mentre que la primera només dona el temps total del procés, la segona és capaç de distingir entre temps en mode usuari i mode sistema. A més, *getrusage* també proporciona informació addicional sobre els recursos consumits per un procés, com la memòria consumida, estadístiques d'entrada / sortida, etc. A la figura 2.5 es pot observar el fragment del codi de Linux, concretament del fitxer *sys/resource.h*, on es defineix l'estructura que retorna la funció, amb tots els seus paràmetres.

De tota manera, per arribar a consultar informació específica del processador, ja sigui el nombre d'instruccions, les fallades de la memòria cau, etc., cal un suport més específic per part d'aquest

```

struct rusage {
    struct timeval ru_utime; /* user time used */
    struct timeval ru_stime; /* system time used */
    long    ru_maxrss;      /* maximum resident set size */
    long    ru_ixrss;      /* integral shared memory size */
    long    ru_idrss;      /* integral unshared data size */
    long    ru_isrss;      /* integral unshared stack size */
    long    ru_minflt;     /* page reclaims */
    long    ru_majflt;     /* page faults */
    long    ru_nswap;      /* swaps */
    long    ru_inblock;    /* block input operations */
    long    ru_oublock;    /* block output operations */
    long    ru_msgsnd;     /* messages sent */
    long    ru_msrvcv;     /* messages received */
    long    ru_nsignals;   /* signals received */
    long    ru_nvcsw;      /* voluntary context switches */
    long    ru_nivcsw;     /* involuntary context switches */
};

```

Figura 2.5: Fragment de sys/resource.h on es descriu l'estructura rusage

component. Els processadors actuals incorporen una sèrie de registres especials anomenats comptadors *hardware*. Cada comptador s'encarrega de comptar un cert esdeveniment del processador com els descrits anteriorment. La diferència entre dues mostres preses a l'inici i al final de l'execució, o en moments particulars de l'execució ens ofereixen el valor de la mètrica en aquell interval de temps.

Per raons de disseny dels processadors, aquests solen tenir un nombre força limitat de registres on guardar aquests comptadors, i disposen d'uns registres especials de control que indiquen quin comptador cal emmagatzemar. També per aquest motiu hi ha certs comptadors que no es poden extreure a la vegada, ja que ocupen el mateix registre. D'aquesta manera, quedaran definits una sèrie de conjunts de comptadors que podran ser extrets alhora.

Tot i això, cada processador és diferent i per tant els comptadors que ofereix i la forma d'accedir-hi pot variar. Amb aquest propòsit va néixer PAPI[BDG<sup>+</sup>00], una eina que defineix una interfície i una metodologia comuna per extreure la informació d'aquests comptadors en la majoria de processadors que hi ha actualment al mercat. Aquest fet fa que la seva utilització sigui fàcilment portable entre diferents arquitectures, i per tant que sigui més fàcil per al programador optimitzar el seu codi. De fet, és la base per la majoria d'eines de mesura de rendiment que hi ha actualment, com les que es veuran a la secció 2.4.

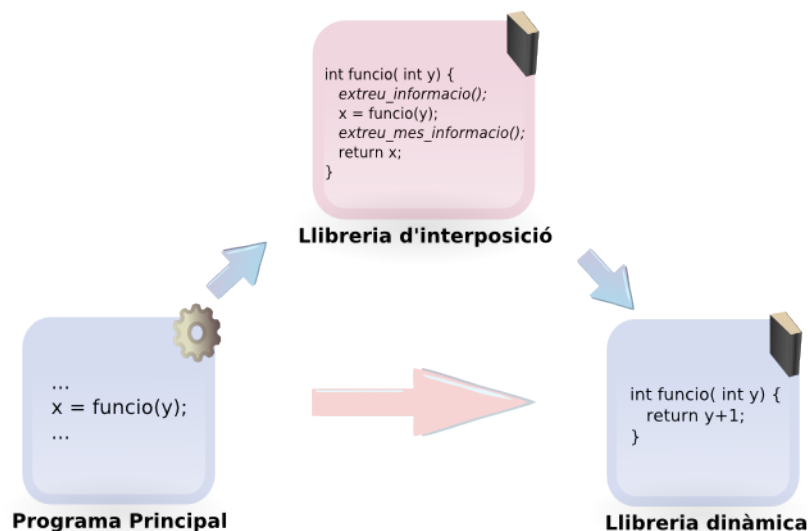


Figura 2.6: Interposició d'una funció per instrumentar-la

PAPI es divideix en dos capes: la part portable i la part dependent de la màquina. Aquesta última, també coneguda com a substrat, cal que sigui implementada per a cada arquitectura i sistema operatiu, ja que és la que haurà de tractar directament amb la informació. En el cas del *kernel* Linux[Tor02], és necessària una extensió per tal que pugui gestionar els comptadors *hardware* correctament.

En qualsevol cas, els mètodes explicats requereixen que es facin crides explícites a les funcions corresponents per tal d'aconseguir els valors per a cada mètrica. Aquest fet, a priori, obliga a modificar el codi de l'aplicació a mesurar, introduint-hi les crides necessàries. Aquest procediment rep el nom d'instrumentació. Per als casos en els que el mateix programador vol fer les mesures, modificar el codi pot ser una opció viable. Però si no es disposa del codi font, o no es vol modificar l'executable original, fa falta alguna estratègia per poder instrumentar de forma dinàmica i en temps d'execució el codi original.

Una forma d'aconseguir aquest objectiu és la interposició de certes crides a funcions que el programa fa i que es troben en una llibreria enllaçada dinàmicament. Com que la funció no està implementada dins del mateix executable, i l'ha d'anar a buscar de forma externa, és el moment idoni per col·locar el codi desitjat d'instrumentació entre la crida de la funció des del programa i la funció original, com es pot observar a la figura 2.6.

Per a que aquest mètode funcioni, cal dir-li al programa que carregui en primer lloc la llibreria d'interposició, abans que les altres. D'aquesta manera, quan el programa faci una crida que no té al seu codi, el primer lloc on mirarà de trobar-la és a la llibreria afegida. Si la funció cridada no és aquí, seguirà buscant a les següents de la llista fins a trobar-la, mentre que si hi és, executarà la

nova versió de la funció, que introduirà el codi necessari per extreure la informació i actuar de pont entre el programa principal i la llibreria original. Lògicament, les funcions han de tenir exactament el mateix prototipus que en l'original.

En Linux es pot aconseguir aquest comportament amb la variable d'entorn LD\_PRELOAD. Per a que funcioni, el seu valor ha de ser el camí cap a la llibreria o llibreries que es volen carregar en primer lloc. El problema d'aquest mètode és precisament que només es pot interposar en crides cap a llibreries dinàmiques, pel que si la crida que es vol instrumentar pertany al mateix fitxer o es d'una llibreria enllaçada estàticament no es pot utilitzar. En aquest cas s'hauria de recórrer a tècniques més sofisticades de reescriptura de la còpia del binari a memòria en temps d'execució.

En el moment de prendre aquestes mesures, cal tenir en compte que qualsevol mètode que permeti mesurar certs paràmetres del comportament d'una aplicació estarà provocant un cert *overhead*, o sobrecàrrega, que penalitzarà en major o menor grau el rendiment de la mateixa. Com més instrumentació s'introdueixi per obtenir més informació o més detall, més codi s'haurà executat, afectant el comportament original del programa. La conseqüència directa serà un augment en el temps d'execució del programa ja que, a banda del codi addicional que s'haurà executat, es podran haver modificat patrons d'accés a memòria, provocant pol·lució de la memòria cau, fallades de pàgina, etc.

És necessari, a més de tenir en compte aquest fet, vetllar per tal que la penalització introduïda no sigui massa gran. Es podria donar el cas que la sobrecàrrega provocada per l'extracció de la informació interferís de tal manera que les dades obtingudes perdessin tot el valor. En qualsevol cas, s'ha de dimensionar la sobrecàrrega per tenir-la en compte a l'hora d'interpretar els resultats aconseguits.

## 2.4 Aplicacions existents

En aquesta secció s'abordan algunes de les aplicacions de mesura i anàlisi de rendiment que existeixen en l'actualitat. Es farà una breu descripció de cadascuna, incidint en les característiques més importants, per tal de veure quina opció és la que s'adapta millor, i quines característiques comunes tenen les diverses alternatives existents. En concret, es distingeixen les aplicacions que simplement extreuen informació donant com a resultat un *profile*<sup>9</sup> o una traça del programa analitzat, i aquelles que, a més, serveixen per fer anàlisi d'aquests resultats.

---

<sup>9</sup>En català, perfil de comportament

### 2.4.1 IBM HPM Toolkit

HPM Toolkit[DeR01] és un conjunt d'eines desenvolupat per IBM per a poder analitzar aplicacions d'alt rendiment i poder detectar els possibles problemes i parts on es podria parar atenció per a millorar el codi. Consta de tres parts diferenciades: una eina que permet arrencar una aplicació i ofereix informació de rendiment al final de la seva execució per a mesures a baix nivell de detall; una llibreria d'instrumentació per a l'estudi amb més resolució i detall de certes parts del programa i per últim una interfície gràfica per a visualitzar i analitzar les dades obtingudes. Aquest conjunt d'eines permet extreure informació tant d'aplicacions convencionals com paral·leles, ja sigui de memòria compartida (OpenMP) o de memòria distribuïda (MPI).

En primer lloc, l'eina més fàcil de cara a l'usuari i que ja proporciona prou informació és `hpmcount`. Aquesta permet donar una idea general del rendiment global d'un programa, donant informació del temps total durant el qual s'ha executat, informació dels comptadors *hardware* així com dades d'utilització d'altres recursos (usant `getrusage`). Per a una granularitat més fina, aquest paquet permet la instrumentació de parts del programa mitjançant la seva llibreria d'instrumentació, però requereixen que l'usuari modifiqui el seu codi afegint les crides a la llibreria de manera estàtica. En aquest sentit també existeix la possibilitat d'utilitzar instrumentació dinàmica utilitzant la API de `Dyninst`[BH00].

Tant una com l'altra poden utilitzar tant els comptadors propis de l'arquitectura IBM PowerPC sobre AIX<sup>10</sup>, per als que s'havia dissenyat originalment, com també comptadors PAPI, cosa que facilita la seva portabilitat a altres sistemes com GNU/Linux sobre una arquitectura Intel. HPM Toolkit permet instrumentar programes escrits en Fortran, C i C++.

Un altre punt a destacar és la utilització de XML com a format per a guardar les dades obtingudes a partir de la llibreria d'instrumentació, ja que possibilita una major independència entre la part d'anàlisi i visualització de la part d'extracció de la informació de rendiment. Aquesta part, anomenada `HPMviz`, és una aplicació que consta d'una interfície gràfica on es pot relacionar les seccions de codi instrumentades, amb les seves etiquetes, amb el temps de duració d'aquesta al mateix temps que es mostra el fragment del programa en qüestió. A més, per a cada secció es pot observar el valor de moltes mètriques, incloent-hi derivades, d'execucions paral·leles o seqüencials.

---

<sup>10</sup>Sistema operatiu propietari de IBM basat en UNIX

## 2.4.2 Papiex

Papiex[Muc05] és una aplicació molt similar a hpmcount, per analitzar informació general del comportament d'una aplicació durant tota la seva execució. Consta d'un executable que permet llançar l'aplicació a analitzar, i l'instrumenta de manera dinàmica mitjançant una llibreria d'interposició, de manera semblant a l'anterior *eina*, pel que tampoc cal recompilar ni modificar el programa original a mesurar. Papiex està desenvolupat entre d'altres per un dels autors de PAPI, i això permet una gran compatibilitat entre totes dues eines.

La informació proporcionada és a nivell de *thread* i de tota la seva execució, que va des del temps total fins a qualsevol mètrica procedent de PAPI, de la crida *getrusage* o d'eines externes com *mpiP* (secció 2.4.4). L'usuari pot especificar quina informació vol extreure, i el programa generarà un informe a la finalització de l'aplicació mesurada.

De la mateixa manera, és capaç de treballar amb aplicacions paral·leles, ja sigui amb *threads*, amb processos MPI o la combinació de les dues opcions. En aquests casos, es genera informació per cada procés i per cada *thread*, per al final generar també un resum de l'execució paral·lela que contindrà mesures estadístiques agregades (mitjana, màxims, mínims, etc). El programa deixarà els fitxers estructurats de manera jeràrquica utilitzant una estructura de directoris.

Papiex s'interposa a les crides de creació i destrucció de processos i *threads*, a més d'algunes poques crides MPI si és necessari, pel que la sobrecàrrega introduïda a l'execució del programa és força baixa. Permet a més, com al cas de HPM Toolkit, definir certes parts del codi on volem mesurar especialment el rendiment mitjançant la mateixa llibreria d'instrumentació, però ara indicant al mateix codi del programa quina secció es vol tractar.

Un avantatge addicional d'aquesta aplicació és que no requereix el seu programa llançador per a portar a terme la instrumentació, ja que la llibreria rep els paràmetres que necessita per al seu funcionament a través de variables d'entorn. De fet, el programa que inicia l'executable a analitzar s'encarrega de carregar aquestes variables d'entorn d'acord amb els valors que se li han passat com a opcions.

## 2.4.3 Perfsuite

Perfsuite[Kuf05] és un conjunt d'eines que permeten extreure informació de rendiment de l'execució d'una aplicació, de manera semblant a les anteriors, oferint diverses utilitats per realitzar aquest anàlisi. Per un costat es troben les llibreries que extrauran la informació, i per l'altre aplicacions que es serviran d'aquestes per facilitar la feina a l'usuari.

En primer lloc, es distingeixen dos llibreries que realitzaran l'extracció. La primera es *libperfsuite*, que extraurà aquella informació que no te a veure amb els comptadors *hardware*, com per exemple l'arquitectura de memòria o el seu ús per part del programa analitzat. Aquesta llibreria és completament independent i només depèn de la presència de la llibreria *glibc*, present en tots els sistemes Linux.

Posteriorment, la llibreria *libpshwpc* s'encarregarà d'extreure la informació de comptadors *hardware* utilitzant PAPI. Aquesta llibreria extreu la informació desitjada al mateix temps que ofereix una petita API per a personalitzar l'extracció modificant el codi del programa. La llibreria recollirà els seus paràmetres de configuració tant de variables d'entorn (com *Papiex*) com d'un fitxer de configuració XML.

Per tal d'evitar utilitzar directament aquestes llibreries es troba l'eina *psrun*. El funcionament d'aquesta aplicació és simplement recollir els paràmetres d'una forma senzilla per al usuari i carregar l'entorn necessari per executar l'aplicació que es passa com a argument, realitzant una precàrrega de les llibreries d'instrumentació.

Com que els resultats obtinguts d'aquestes mesures es generen en un fitxer XML, el paquet també ofereix un programa per convertir aquesta sortida en un informe més entenedor. *Psprocess* és l'encarregat de dur a terme aquesta tasca. El resultat serà un arxiu de text amb les dades obtingudes i organitzades de manera més apropiada.

*Perfsuite* permet realitzar anàlisi d'aplicacions tradicionals amb un sol *thread* o amb diversos *threads* POSIX. OpenMP està suportat indirectament, ja que OpenMP es construeix sobre els *threads* POSIX. El que no té un suport explícit és MPI, ja que tot l'anàlisi es fa a nivell de procés.

#### 2.4.4 MpiP

Aquesta eina[VC05] ofereix a l'usuari la possibilitat de fer un perfil de comportament (*profile*) de la seva aplicació MPI. En aquest cas no s'extreu cap tipus d'informació de comptadors *hardware*, sinó que més aviat dóna una idea de com s'ha executat el programa des del punt de vista paral·lel MPI. El resultat obtingut recorda al que dóna GNU *profile*, ja que recull quant de temps es consumeix en cada grup de crides MPI, o des d'on es realitzen (mostrant també el traçat invers, o *backtrace*). També és útil per conèixer la quantitat d'informació que cada procés ha enviat i rebut a través de MPI cap als altres processos.

A l'hora d'utilitzar-lo, el procediment estàndard és enllaçar la llibreria *mpiP* al moment de compilar el programa, encara que també permet precarregar la llibreria en temps d'execució per tal de no tornar a compilar l'aplicació.

### 2.4.5 Vampir

Vampir[NAW<sup>+</sup>96] és una eina per a la visualització i l'anàlisi del rendiment de programes paral·lels MPI. A diferència de les anteriors, el seu funcionament es basa en l'extracció de traces del programa paral·lel on contindrà informació de certs paràmetres i esdeveniments que han ocorregut durant tota la seva execució. El paquet vampir consta de dos parts principals: el tracejador i el visualitzador.

En primer lloc, per extreure les traces s'utilitza vampirtrace, una llibreria que interceptarà les crides MPI i algunes altres per a generar la informació. El programa a analitzar s'haurà d'enllaçar amb aquesta nova llibreria, i posteriorment executar-lo.

Les traces generades es podran carregar amb el visualitzador de vampir, que serà capaç de generar gràfics sobre les dades obtingudes com línies temporals, patrons de comunicacions MPI, etc. Això permet a l'usuari analitzar d'una forma visual com treballa la seva aplicació, detectant els problemes que pugui tenir.

Degut al gran nivell de detall que requereix aquest anàlisi, les traces solen ser molt més grans que els fitxers generats per altres eines descrites anteriorment, ja que recullen informació del que ha passat durant tota l'execució, i el seu tractament no és tan lleuger. Per altra banda, aquest tipus d'eines permet un anàlisi molt complet de l'aplicació, oferint tot tipus de possibilitats a l'hora de generar gràfics, estadístiques i detectar seccions problemàtiques del codi.

Tot plegat requereix la intervenció de l'usuari en totes les fases, donat que cal tornar a enllaçar el programa amb la nova llibreria d'instrumentació, per després amb les traces generades fer un anàlisi pel que cal una certa base de coneixements.

### 2.4.6 Paraver

Seguint amb les eines de traceig, existeix una eina desenvolupada des del CEPBA<sup>11</sup>, actualment integrat al BSC-CNS que s'anomena paraver[PLC<sup>+</sup>95]. És una aplicació semblant a l'anterior, que permet extreure traces de l'execució d'un cert programa, que contindran informació i dades dels valors de determinats paràmetres i esdeveniments.

Com vampir, paraver ofereix una llibreria d'instrumentació contra la que s'haurà d'enllaçar el programa MPI o OpenMP que es vol analitzar (mpitrace o omptrace, segons el cas). La llibreria es podrà enllaçar ja sigui en el moment de compilar l'aplicació o bé seguint el mecanisme de

---

<sup>11</sup>Centre Europeu de Paral·lelisme de Barcelona, associat a la UPC

precàrrega de llibreries. Aquesta interceptarà certes crides MPI per poder extreure en aquests moments la informació desitjada i escriure-la a la traça. Disposa d'un arxiu de configuració xml on s'especificaran les opcions de la llibreria, tals com quina informació extreure o determinats aspectes de les traces generades.

Un cop executada l'aplicació instrumentada, es generen tants fitxers de traces com processos MPI, que caldrà unir en un únic fitxer carregable per l'eina de visualització. Paraver també ofereix una eina per juntar la traça (mpi2prv), que generarà un fitxer amb extensió ".prv" que contindrà la informació de tots els processos.

Amb el fitxer ".prv", s'obrirà el visualitzador paraver, que permetrà tot tipus d'anàlisi visual del comportament del programa, disposant d'una gran quantitat de funcionalitats per retallar o fer zoom sobre certes seccions de l'execució, analitzar dades en 2 o 3 dimensions, filtrar certs esdeveniments, així com fer una correlació entre diferents gràfics i comparar traces de diferents execucions. Segons el programa analitzat i la quantitat d'esdeveniments que es volen observar, les mides dels fitxers de traces poden ser molt grans, tot i que per solucionar-ho el programa ofereix la possibilitat de filtrar o tallar la traça per obtenir una de més manejable.

Paraver és un sistema molt complet, però alhora complex de fer servir. L'usuari que realitza l'anàlisi ha de conèixer bé la metodologia de treball d'aquesta eina, així com les possibilitats que li ofereix. La flexibilitat que ofereix el programa penalitza la seva facilitat d'ús, pel que cal una certa experiència amb el programa per aprofitar al màxim les seves possibilitats.

### 2.4.7 Tau

Tau (Tuning and Analysis Utilities)[SM06] es pot considerar un *framework* o entorn de treball per a l'anàlisi de rendiment d'aplicacions, ja que engloba moltes altres eines i les integra en un mateix sistema. D'aquesta manera, permet per una part extreure dades del tipus perfil de les aplicacions (*profiles* com el de GNU, gprof), així com dades de tipus traça, obtenint-les des de la instrumentació del codi. Aquestes poden provenir de diferents fonts, de manera que tau compta amb una capa de traducció i emmagatzemament de la informació que permetrà la seva visualització amb diferents eines de manera fàcil.

De la part de l'extracció de les dades, ofereix diferents possibilitats com la instrumentació a nivell de codi, a nivell de compilador o dinàmicament al binari utilitzant DynInst. També permet instrumentar programes escrits en llenguatges interpretats (com Python), així com aquells basats en màquines virtuals (com Java).

Per a emmagatzemar les dades, utilitza l'anomenat Performance Data Management Framework (PerfDMF), sobre el qual treballaran les eines d'anàlisi. Consisteix en una base de dades SQL on es recull la informació extreta de les diferents fonts, que prèviament haurà estat traduïda per a poder-la emmagatzemar. Aquesta base de dades esta orientada a la realització dels anomenats experiments, que són cada execució dels programes que es volen analitzar. PerfDMF compta amb una API per accedir a aquesta base de dades, i que permet extreure la informació en múltiples formats, com XML. Les eines d'anàlisi poden utilitzar la base de dades directament, o utilitzar aquesta API per no haver de tractar SQL explícitament.

Per a l'anàlisi i visualització, a més de poder convertir les traces obtingudes a formats coneguts per altres eines com Vampir o Paraver, es pot utilitzar ParaProf, una interfície gràfica que genera les gràfiques, estadístiques, o fins i tot grafs de crides del programa. Aquesta aplicació pot accedir a la base de dades per extreure la informació, i permet també un anàlisi força flexible.

Tau és un conjunt d'eines que permeten extreure i analitzar el rendiment d'aplicacions a molts nivells, i que ahora agrupa en un mateix paquet utilitats per analitzar tant el seu perfil com el seu rendiment. Tanmateix, això també implica que pot produir una sobrecàrrega considerable a l'execució normal de les aplicacions. A més, requereix la intervenció de l'usuari en diversos punts del procés, ja que està pensat fonamentalment per l'anàlisi d'una aplicació o aplicacions d'un mateix usuari.

### 2.4.8 PerfMiner

PerfMiner[MAD<sup>+</sup>05] és una eina que permet extreure informació de rendiment de manera transparent i global de totes les aplicacions executades en un o més clústers. A diferència de les anteriors, s'extreu informació de totes les aplicacions que estan en execució i es recull per emmagatzemar-la tota junta en una base de dades, sense que l'usuari hagi de modificar el seu entorn de treball. Aquesta infraestructura està desenvolupada per als clústers del PDC (Center for Parallel Computers), a Suècia, per un dels autors de PAPI.

Al nivell més baix, utilitza papiex (secció 2.4.2) per a l'extracció de la informació de rendiment. Es crea una *shell* especial per carregar l'entorn necessari per a l'obtenció de les dades abans de cada programa que l'usuari executi. En aquest punt s'executa papiex, a l'hora que es recull informació del sistema de cues corresponent (si n'hi ha) com a complement. En els sistemes originals del PDC utilitzen un sistema de cues anomenat Easy, i el sistema està pensat per integrar-se amb aquest planificador, que és bastant limitat.

Els fitxers amb les dades resultants seran recollits posteriorment per inserir tota la seva informació dins una base de dades. PerfMiner utilitza PostgreSQL com a sistema gestor de bases de dades tot i que és possible la seva adaptació a altres sistemes gestors. El disseny de la base de dades s'orienta al clúster i no a una execució o aplicació, per tal de poder guardar informació de totes les aplicacions que s'executen. Ofereix diferents nivells de resolució per tal de poder després generar dades agregades fàcilment.

Per últim, PerfMiner compta amb una interfície web per a poder consultar la informació que conté la base de dades. Permet analitzar d'una manera visual la informació obtinguda, generant taules i gràfiques de mètriques a diferents resolucions (des de clúster fins a *thread*). L'usuari que consulta aquest web pot seleccionar la mètrica o mètriques desitjades d'una llista, filtrar per usuaris o treballs, etc.

Degut a que és un sistema que pretén extreure informació de moltes execucions, la informació que s'obté de cadascuna no té el nivell de detall d'altres que ja s'han tractat. Cal tenir en compte que el volum de dades tot i tenir només informació global de l'execució de cada *thread* és molt gran, i resulta difícil de tractar tant a l'hora d'emmagatzemar la informació com a l'hora de consultar-la. A més, aquest sistema està pensat i implementat per clústers petits amb una configuració concreta, pel que la portabilitat a d'altres sistemes pot requerir modificacions importants.



## Desenvolupament del projecte

Feta ja la introducció necessària, aquest capítol es centra en el del projecte realitzat. S'abordan els aspectes més rellevants del seu desenvolupament en les diferents parts que el componen, tot indicant els requisits específics de cada part, el punt de partida, el desenvolupament realitzat i les decisions de disseny preses en cada punt. De tota manera, cal primer situar el punt de partida del projecte i establir les bases i els principis que regiran el projecte abans d'entrar en detall a cada paquet de treball.

### **3.1 Punt de partida**

A continuació s'exposen els requisits generals que haurà de tenir el sistema, per després valorar l'eina utilitzada com a base per a començar a treballar justificant la seva elecció. Per últim, es recordarà la divisió del treball realitzada per aquest projecte, d'acord amb les diverses parts que formen el sistema.

#### **3.1.1 Requisits generals**

En el moment de buscar una solució al problema proposat, sorgeixen una sèrie de requisits fonamentals que el sistema que es construirà haurà de complir. De les motivacions i els objectius que s'han marcat per a la realització d'aquest projecte (seccions 1.1 i 1.2) se'n deriven aquests requisits.

Com s'ha comentat en seccions anteriors, es vol tenir un sistema capaç d'extreure informació a nivell global del rendiment que tenen les aplicacions executades per tots els usuaris en un cert supercomputador o supercomputadors, ja que això possibilitaria conèixer millor l'ús que els usuaris fan dels recursos, i el comportament de les mateixes aplicacions executades amb circumstàncies diferents.

La *transparència* és una de les fites a aconseguir en aquest projecte. L'usuari no s'hauria de preocupar per la recollida de la informació dels seus treballs. La forma com s'extreu la informació, com es recull o com s'emmagatzema haurien de ser totalment independents de l'usuari final. Fins i tot, podria donar-se el cas d'ignorar completament aquest fet, i seguir treballant de la mateixa manera i amb el mateix entorn que tenia prèviament.

Un altre punt important és garantir la *mínima interferència* amb l'execució normal de l'aplicació, intentant que la penalització introduïda amb les eines de mesura sigui la menor possible. Cal que tots els processos introduïts no afectin significativament el rendiment tant de l'aplicació com del sistema en general. Relacionat amb aquest concepte estaria l'*eficiència* del sistema, ja que l'entorn on s'haurà d'integrar l'eina és un entorn de computació d'alt rendiment que s'ha de destorbar el mínim possible.

En aquest sentit, donat que el sistema haurà de ser capaç de treballar en grans clústers, ens interessa que sigui *escalable* a mides de l'ordre de milers de nodes on s'executaran milers de treballs de mides variables. L'escalabilitat s'haurà de tenir en compte tant en la part de la recollida de la informació com a la del seu emmagatzemament.

### 3.1.2 Elecció de l'eina: PerfMiner

Finalment, PerfMiner (secció 2.4.8) serà el sistema del qual es partirà per realitzar aquest projecte. Després de valorar els pros i contres de les diverses opcions vistes a la secció 2.4, s'ha escollit aquesta ja que és la que més s'adapta als objectius i els requisits plantejats.

PerfMiner és l'única solució que no es concentra en una aplicació o aplicacions en concret, sinó que va més enllà i ofereix una informació més general del rendiment i de l'ús del sistema complet, ja que recull dades de tot el que s'executa a cada moment. Com que per a l'obtenció d'aquestes dades es fa servir papiex, el resultat de caràcter general, sense oferir el detall d'una traça, per a tenir un volum de dades tractable tant a nivell de la seva recollida com del seu emmagatzemament.

A més, PerfMiner parteix del principi de la transparència envers l'usuari, que no ha de realitzar l'extracció explícitament, sinó que és el sistema qui s'encarrega d'aquesta tasca. La idea d'oferir

una interfície web per a realitzar la visualització i l'anàlisi dels resultats és també molt interessant, ja que facilita l'accés des de qualsevol terminal sense necessitat d'instal·lar cap tipus de programa, com passa amb la resta d'eines.

No obstant, aquesta solució no pot competir en flexibilitat ni potència d'anàlisi amb d'altres. ja que precisament la seva arquitectura fa que el nivell de detall en la informació i les possibilitats del seu anàlisi siguin limitades. Cal també remarcar que PerfMiner està pensat per una configuració de sistema concreta, ja que requereix una integració considerable dins l'entorn d'execució. Aquest fet és la causa de que la portabilitat cap a clústers amb una configuració diferent, i sobretot molt més grans, sigui més complexa del que pot semblar en un principi.

Caldrà doncs partir del programa original per adaptar-lo per tal que pugui funcionar de forma eficient en un supercomputador de dimensions de centenars o milers de processadors. En certs casos caldrà un complet re-disseny d'algun component per assegurar que els requisits marcats es compleixen.

Cap eina de les tractades a la secció 2.4 s'ajusta tant als objectius marcats com PerfMiner. Per començar, cap s'integra dins del sistema per oferir la transparència desitjada envers l'usuari. Tot i així, es podrien haver utilitzat altres eines d'obtenció de la informació com les que ofereixen HPM Toolkit o Perfsuite. Aquestes també proporcionen una informació general sobre el programa, i l'instrumenten de manera dinàmica. A més, generen informació resumida del rendiment de l'execució lluny de les mides de fitxers de traça com els que generarien eines com Paraver, Vampir o Tau.

Exceptuant Tau, cap de les eines vistes està orientada a treballar sobre una base de dades on desar-hi tota la informació recollida. Estan pensades per analitzar el rendiment d'una aplicació en concret, pel que els fitxers generats amb la informació o les traces ja constitueixen la base de l'anàlisi. Tau, però, té l'inconvenient de ser un sistema massa gran i també molt orientat a execucions concretes, cosa que faria difícil analitzar dades a nivell de clúster o d'usuari en general.

En resum, les raons de l'elecció de Perfminer han estat la seva lleugeresa a l'hora d'obtenir la informació, al mateix temps que ho fa de manera transparent. El seu enfoc per l'anàlisi global de les dades obtingudes en clústers de computació s'ajusta prou bé a la idea inicial amb la que s'havia concebut aquest projecte.

## 3.2 Estructura general del sistema

Tot seguit es presenta l'arquitectura general del sistema a instal·lar, en la que intervenen diversos elements que portaran a terme determinades tasques. A la figura 3.1 es pot observar gràficament un esquema de la infraestructura desitjada. Prenent com a referència l'estructura de PerfMiner, el sistema es divideix en les següents parts, i que seran detallades en seccions posteriors:

1. Obtenció de la informació de rendiment.
2. Integració en l'entorn d'execució.
3. Recollida de la informació.
4. Emmagatzematge.
5. Anàlisi i visualització.

El funcionament del sistema és el següent: l'usuari del supercomputador enviarà un treball al sistema de cues per a que s'executi un cert càlcul de la manera habitual (veure secció 2.2.4). Quan el treball comenci a executar-se, s'hauran fet les modificacions necessàries en l'entorn d'execució per tal d'instrumentar de manera dinàmica tots els processos que entrin en execució mentre el treball estigui actiu. D'aquesta manera, s'obté informació de tot el que s'ha executat, siguin aplicacions seqüencials o paral·leles. La informació generada s'anirà desant en un directori temporal, a l'espera que el treball acabi.

Quan un treball seqüencial acaba, el processador que tenia assignat s'encarregarà dels fitxers temporals generats. En canvi, quan es tracta d'un treball que executa una aplicació paral·lela, només un dels processadors assignats realitzarà aquesta feina. Concretament, serà el processador identificat com el primer (el que sigui el número 0 dins el treball), l'únic que treballarà amb els fitxers temporals.

Des d'aquest punt s'enviarà a través de la xarxa les dades recollides cap a uns servidors que faran de magatzem temporal d'aquestes dades. Es pretén que puguin coexistir més d'un servidor d'aquest tipus per tal de millorar tant la tolerància a fallades com el balanceig de la càrrega que es genera en aquest procediment.

En aquest moment les dades estan esperant a ser desades definitivament a la base de dades. El pas següent és precisament aquest: periòdicament, els servidors de recollida aniran bolcant a la base de dades tota la informació que han acumulat. El sistema està pensat de manera que aquesta base

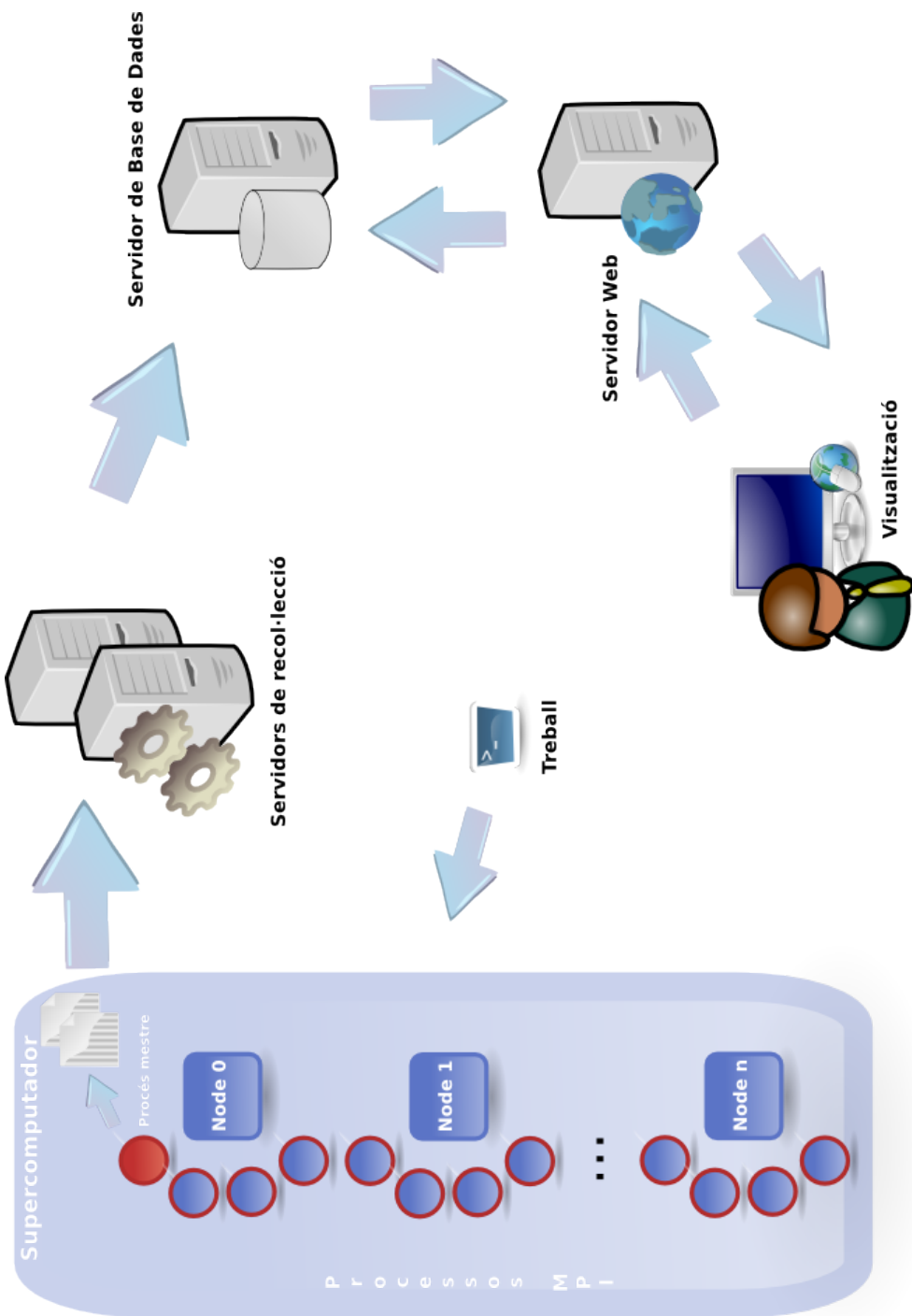


Figura 3.1: Estructura general del sistema a desenvolupar

de dades pugui estar en un nou servidor, separant així els diferents serveis i millorant el rendiment global evitant sobrecarregar innecessàriament algun servidor.

És en aquest instant quan el rendiment del treball enviat per l'usuari pot ser visualitzat a través d'una interfície web. Novament, és desitjable tenir el servei web allotjat en un altre servidor diferent dels anteriors. A partir d'aquí es podrà analitzar el rendiment del treball enviat juntament amb els altres, obtenint informacions de rendiment a nivell global.

En les properes seccions es fa una explicació més específica de cadascuna de les parts que componen el sistema, entrant en detalls de la implementació i la justificació de les decisions de disseny preses que han donat com a resultat l'arquitectura presentada, tot seguint l'ordre lògic suggerit per l'esquema presentat.

### 3.3 Extracció de la informació de rendiment

Prenent com a referència PerfMiner, com ja s'ha anunciat en seccions anteriors, l'eina utilitzada per extreure la informació de rendiment és Papiex. En l'actualitat, Papiex està en actiu desenvolupament per afegir-hi noves funcionalitats i major compatibilitat amb gran nombre de plataformes. Com s'ha comentat a la secció 2.4.2, les seves característiques s'adapten força bé a les necessitats que cal cobrir en aquest punt. De tota manera, ha estat necessari afegir noves funcionalitats per tal d'adaptar completament l'aplicació. A continuació es detalla el funcionament de Papiex per posteriorment descriure el que ha calgut modificar.

#### 3.3.1 Anàlisi del funcionament de Papiex

Papiex permet extreure informació general de l'execució d'una aplicació fins a una resolució de *thread*. Consta bàsicament d'una llibreria d'instrumentació, que es la que fa la feina d'extracció, i un llançador de l'aplicació a analitzar. Tot i que permet afegir instrumentació afegint certes crides dins el codi del programa cap a la llibreria, l'ús que se'n farà serà només el de la instrumentació dinàmica en temps d'execució. Per fer-ho, instrumenta les rutines de creació i destrucció tant de processos com de *threads*, utilitzant una llibreria de suport anomenada monitor.

Monitor intercepta crides de la llibreria glibc que es fan des de l'aplicació. Cal notar que per tal que funcioni, el programa ha d'estar enllaçat dinàmicament amb aquesta llibreria. Altrament, aquestes rutines estarien dins de l'objecte executable, provocant que el programa no anés a buscar el símbol a l'exterior i impossibilitant la intercepció. Un cop realitzada la interposició, monitor

executa una crida (*callback*) a una funció definida a Monitor però que implementarà l'eina que l'utilitza (en aquest cas Papiex). D'aquesta manera ofereix una forma fàcil d'accedir a certs punts claus de l'execució d'un programa i permetent que s'executi un determinat codi, que serà el de instrumentació. Entre les rutines interceptades trobem, per la part de processos `libc_start_main`, `fork`, `exec`, `exit`, etc. i per la part de *threads* funcions com `pthread_create` o `pthread_mutex_lock`.

Tant la llibreria Monitor com la de Papiex han de ser carregades en primer lloc per l'aplicació mitjançant la variable `LD_PRELOAD` a Linux. A més de les possibilitats ofertes per Monitor, Papiex també intercepta altres crides per a mesurar l'ús de recursos com l'entrada sortida (`open`, `close`, etc) o crides MPI per tal de generar estadístiques també d'aquest aspecte. D'aquestes últimes no s'intercepten ni molt menys totes les existents, d'una banda per evitar crear massa sobrecàrrega i de l'altra perquè la informació que es dona és de caire general (només intercepta crides com `MPI_Init`, `MPI_Finalize` i poques més). A més, la llibreria està feta de tal manera que no cal enllaçar-la amb la llibreria MPI si aquesta és una versió MPICH (sí que cal en el cas d'altres implementacions com OpenMPI).

Papiex no instrumenta funcions d'usuari, a menys que es facin certes crides des del mateix codi a analitzar (cosa que no s'aplica al cas d'ús d'aquest projecte), i només escriu els resultats a l'acabar l'execució. Papiex té dos modes de generar la sortida, depenent de la quantitat d'informació i el format que es vulgui com a sortida. Per al sistema que estem construint, utilitzarem el mode estàndard ja que proporciona informació addicional necessària per a poder interpretar els resultats correctament i d'una manera més senzilla.

A més, es pot escollir si escriure el resultat per la sortida d'error estàndard, en fitxers, o tots dos. Per a PerfMiner, i tenint en ment la transparència desitjada envers l'usuari, la millor opció és optar per només generar la sortida en forma de fitxers, que a més quedaran emmagatzemats en un directori temporal separat del directori on l'usuari està realitzant la seva execució. D'aquesta manera s'aconsegueix interferir el menys possible en la rutina de treball de l'usuari, ja que no es contaminarà la sortida generada pel programa ni el directori on treballa.

Els fitxers generats s'organitzen de manera jeràrquica. Així, un procés que formi part d'una aplicació no paral·lela, generarà per si sol un fitxer amb la informació pertinent. Aquest fitxer portarà com a nom una combinació de paràmetres com el nom de l'executable o el seu *pid*. Per contra, una aplicació paral·lela (ja sigui MPI o amb *threads*), generarà els seus fitxers dins d'un nou sub-directori (amb la mateixa política de noms anterior) pertanyent a l'execució paral·lela. A dins, els fitxers aniran numerats segons el seu identificador dins l'entorn paral·lel, i també es generarà un fitxer extra d'aquella execució amb un resum de tots els seus components.

Cal destacar que en el context d'una aplicació paral·lela, l'escriptura dels fitxers també es fa en paral·lel. Això és, cada procés escriurà la seva informació i per tant es fa necessari que tots els processos tinguin la mateixa visió del sistema de fitxers on s'escriu, com seria el cas de GPFS (veure secció 2.2.3).

A la figura 3.2 es pot observar un exemple de sortida de Papiex corresponent a l'execució d'un programa paral·lel MPI. Concretament, la sortida és d'un dels processos MPI. El fitxer generat ocupa de l'ordre de 1 a 10 KB. Com a informació rellevant (a més de les mètriques), trobem l'executable i els seus arguments, el node on s'ha executat, el *pid* i *ppid* del procés i els instants de començament i finalització del mateix. Es poden veure també les opcions del programa, que han estat passades mitjançant la variable d'entorn PAPIEX\_OPTIONS, explicada tot seguit.

El llançador d'aplicacions no és més que una manera fàcil per a l'usuari de carregar l'entorn adequat per al seu ús. Les opcions passades per línia de comandes es recolliran abans d'arrencar l'aplicació, per tot seguit transformar-les en variables d'entorn:

- PAPIEX\_OUTPUT: Aquí es definirà on escriurà Papiex els fitxers amb els resultats de rendiment.
- PAPIEX\_OPTIONS: És en aquesta variable on, entre altres coses, s'especifica quina informació s'obindrà del programa instrumentat, i com serà la sortida generada.

A més d'aquestes, el llançador haurà d'exportar les llibreries que s'interposaran i on es poden trobar. Aquestes variables són LD\_PRELOAD i LD\_LIBRARY\_PATH, respectivament. El fet que tot funcioni a través de variables d'entorn, possibilita que no calgui utilitzar el llançador per a poder extreure la informació. N'hi ha prou amb carregar-les manualment amb els valors adients abans d'executar el programa.

### 3.3.2 On escriure la informació obtinguda

Com s'ha apuntat en seccions anteriors, els fitxers generats amb la informació de rendiment s'han d'emmagatzemar temporalment en algun indret abans de ser tractats, i l'usuari i el seu entorn de treball habitual no han de veure's afectats. Es necessita doncs un directori temporal on Papiex escrigui els fitxers generats. A més, és desitjable tenir agrupats tots els fitxers pertanyents al mateix treball en un mateix directori separat dels produïts en altres treballs, ja que això facilitarà el tractament posterior.

```

papiex version           : 0.99rc9
Executable               : /home/Operacions/xabellan/paral_MPI/LUB
Arguments                :
Processor                : PowerPC 970
Clockrate                : 2194.624512
Hostname                 : nodotest1
Options                  : DIR,WRITE_ONLY,MEMORY,
                        NO_SCIENTIFIC,PAPI_TOT_CYC,PAPI_TOT_INS

Domain                   : User
Parent process id        : 19959
Process id               : 19961
Start                    : Tue Jun 10 14:37:53 2008
Finish                   : Tue Jun 10 14:37:56 2008
Derived Metrics:
IPC .....                1.40
CPU Utilization .....    0.32
I/O Cycles % .....       6.30
Cycles .....             1949764209
Instructions Completed ..... 2722753363
Mem virtual peak .....   0
Mem resident peak .....  0
Mem text .....           12
Mem library .....        5484
Mem heap .....           91160
Mem stack .....          540
Mem shared .....         5820
Mem locked .....         0
Real usecs .....         2781436
Real cycles .....        6102451278
Proc usecs .....         1734823
Proc cycles .....        3806178222
I/O cycles .....         384596058
PAPI_TOT_CYC .....       1949764209
PAPI_TOT_INS .....       2722753363

```

Figura 3.2: Exemple d'arxiu generat per Papiex

En aquest punt es planteja la qüestió de quin sistema de fitxers s'utilitzarà per allotjar aquest directori temporal. En una configuració com la de MareNostrum, apareixen les opcions d'utilitzar GPFS o els discos locals de cada node (veure secció 2.2.3). Cada opció té les seves particularitats, que detallem a continuació.

Decantar-se per GPFS sembla l'opció més senzilla, ja que és un sistema de fitxers compartit i vist de la mateixa manera des de qualsevol punt del supercomputador. Això permet que la informació generada per una aplicació paral·lela s'escriui tota al mateix directori temporal d'aquest sistema de fitxers. Tal com s'ha vist en l'apartat anterior, Papiex està pensat per treballar amb aquesta premissa. Cal notar que les aplicacions seqüencials no representen en aquest sentit cap problema, ja que s'executen en un sol node.

Per altra banda, la forma d'escriure els fitxers de Papiex no és la més adequada per treballar amb GPFS. És fàcil veure que per una execució MPI de gran envergadura es generaran gran quantitat de fitxers al mateix temps i al mateix directori. Per exemple, si s'executa Papiex per treure informació d'una execució MPI amb 1000 processadors, al acabar es generaran 1000 fitxers a la vegada en el directori temporal definit, que serà el mateix per a tots ells. GPFS haurà de gestionar aquesta escriptura paral·lela amb cura, ja que per crear el fitxer cada procés haurà d'adquirir el dret d'escriptura sobre el directori en exclusiva. Mentre un procés creï el seu fitxer els altres hauran d'esperar, provocant contenció.

A la sobrecàrrega introduïda a GPFS per aquesta execució se li ha de sumar la resta de treballs que en un determinat moment poden estar generant també els seus fitxers de Papiex. Amb tot, aquesta solució pot entrar en conflicte amb un dels requisits marcats per al sistema: la mínima interferència. Eventualment, aquestes pràctiques aporten una càrrega de treball extra al sistema de fitxers que poden derivar, si el sistema està essent activament utilitzat, en una degradació de l'entrada/sortida de no només el mateix treball sinó també els altres que coexisteixen al supercomputador. Fins i tot, donat el cas, es podria arribar a col·lapsar completament el sistema.

Aquest problema se soluciona amb la opció dels sistemes de fitxers locals. El fet de que cada node dels que han de generar els fitxers de sortida utilitzi el seu propi sistema de fitxers possibilita que es redueixi dràsticament la contenció en l'escriptura de fitxers. En aquest moment en cada node només s'escriuran a la vegada tants fitxers com processos estiguin executant en aquell node (com a màxim 4, en el cas de MareNostrum). A més, la càrrega de la generació dels fitxers es trasllada des dels servidors de disc (en el cas de GPFS) fins a cadascun dels nodes involucrats, de manera que es produeix una escriptura més distribuïda.

La part negativa de la solució dels discos locals apareix precisament en aplicacions paral·leles. En el cas d'aplicacions seqüencials els fitxers generats queden allotjats tots al mateix node i per tant el

treball de recuperar-los és senzill. En canvi, una aplicació paral·lela deixarà fitxers amb informació en cada node on s'ha executat, de manera que la recol·lecció es fa molt més difícil. Hi ha doncs dos possibles solucions per aquesta qüestió, i que tenen a veure com es farà la recol·lecció de la informació.

La primera solució passa per complicar la recol·lecció. Tots els nodes que han participat a l'execució prendrien part activament en la recol·lecció en el moment que s'hagués d'enviar la informació cap al servidor de recollida. El problema d'aquesta opció és la càrrega que es generarà en aquest servidor, ja que haurà d'atendre milers de connexions simultànies (una per cada node). A més, es perd la unitat de la informació, de manera que es complica també la identificació de cada part d'informació recollida.

Un altre camí és realitzar una estructura jeràrquica de recollida i simular el comportament d'un sistema de fitxers compartit. La informació generada a cada node és enviada cap a un d'ells, que actuarà com a node mestre. Posteriorment, aquest últim escriurà tots els fitxers al mateix directori del sistema de fitxers local. En aquest escenari només el node mestre s'haurà de comunicar amb el servidor de recol·lecció, reduint el nombre de connexions i càrrega d'aquest servidor.

Com que l'escriptura es duria a terme en un context local, cap altre treball ni cap altre punt del supercomputador es veuria afectat. Com a complement, en una màquina com MareNostrum es pot utilitzar la infraestructura Myrinet per a l'enviament de la informació des dels nodes treballadors fins al node mestre. D'aquesta manera tampoc la xarxa que sosté el sistema de fitxers es veuria afectada, i per la xarxa Myrinet no suposaria una sobrecàrrega notable, ja que només suporta les comunicacions (veure secció 2.2.2).

### 3.3.3 Adaptació

Papiex s'adapta molt bé als requeriments proposats, ja que és capaç de treure informació general de pràcticament qualsevol aplicació, ja sigui paral·lela o seqüencial. A més, suporta tant paral·lelisme a nivell de memòria compartida (*threads*) com de distribuïda (MPI), donant detalls de l'execució de cada procés i cada *thread*. Tot i això, han estat necessaris alguns ajustaments i modificacions a l'aplicació per assolir el nivell de funcionalitat esperat, que queden explicades tot seguit.

El primer que es troba a faltar és informació d'identificació de les execucions paral·leles MPI, com per exemple l'identificador del procés dins de l'execució MPI (*rank*). Si bé és cert que aquest valor dona nom a cada fitxer generat, aquesta informació no apareix en el propi fitxer. Aquest fet és poc important si l'anàlisi és manual, però pot ser una mica incòmode si es fa un tractament

automàtic. Seria molt més fàcil tenir també aquesta informació escrita dins del fitxer, que al ser llegit ja contindria tota la informació necessària per identificar cada procés. S'ha afegit doncs aquesta opció per tal que s'imprimeixi aquesta dada en el cas de tractar-se d'una execució MPI juntament amb les altres dades que identifiquen el procés.

Un cop distingits els processos dins d'una execució MPI, cal identificar també les execucions MPI dins d'un mateix treball. Com que dins d'un treball es poden dur a terme diverses execucions paral·leles, es podria afegir també la informació necessària per a que, mirant un fitxer dels que han estat generats, es pogués dir al moment a quina execució pertany, de la mateixa manera que es fa amb el *rank*.

Altre cop, això pren sentit quan es treballa sota un entorn automàtic i de treballs en cues. SLURM proporciona una variable d'entorn que identifica cada execució paral·lela dins d'un *job* (anomenada *step*, o pas en català). La variable en qüestió és SLURM\_STEPID. S'ha afegit doncs la possibilitat que en aquestes circumstàncies, s'imprimeixi el valor d'aquesta variable.

Per últim, queda el tema més delicat: on s'escriu la informació generada. En la secció anterior s'ha tractat amb detall el problema, i les possibles solucions. Com s'ha vist, Papiex està pensat per a treballar sota un mateix sistema de fitxers quan cada procés d'una aplicació paral·lela escriu la seva informació de rendiment. Per tant, per aquesta solució no cal realitzar cap modificació. En canvi, sí que cal treballar per a poder oferir la opció d'utilitzar sistemes de fitxers separats i no compartits. Breument, la solució passava per que tots els processos enviessin la seva informació cap un procés mestre per tal que tots els fitxers acabin escrits en el sistema de fitxers visible per aquest.

Com que l'escriptura del fitxer es duu a terme en diversos punts del codi de papiex, es fa difícil definir un punt des del qual enviar tota la informació sense arribar a modificar gran quantitat de codi. S'ha optat doncs per deixar que cada procés generi el seu fitxer, i abans d'acabar i sortir, llegir-lo per enviar tot el que contingui i esborrar-lo. Aquesta solució és la més adient en aquest cas, ja que permet introduir la nova opció de manera més fàcil, i també redueix les possibilitats de que es produeixin errors en el programa.

Fent-ho així també afavorim la mantenibilitat del codi, ja que mentre no s'integri en la versió oficial, s'hauria d'adaptar per a cada versió nova que aparegués, a més de afectar el menys possible en el funcionament habitual de Papiex. També per aquesta raó, s'ha implementat aquesta funcionalitat com a una opció addicional més de Papiex (afegint el valor NOT\_SHARED\_GATHER a la variable d'entorn PAPIEX\_OPTIONS). D'aquesta manera, és fàcil canviar el mode de funcionament per treballar amb sistema de fitxers compartit o separat i fer-ho en temps d'execució del programa.

L'escenari ideal per a realitzar l'enviament és la xarxa Myrinet, i no hi ha millor manera d'utilitzar-la que fent servir MPI. A més d'oferir eficiència i facilitat d'implementació, MPI també dóna al codi un alt grau de portabilitat, ja que en cas de no tenir Myrinet, l'enviament es realitzaria igualment a través de la xarxa existent. La clau per a realitzar aquesta recollida és la funció `MPI_Gatherv`, que permet al procés mestre recollir d'una tacada tots els fitxers que li envien la resta.

Complementàriament, es recullen les mides de cada un dels fitxers de cada procés utilitzant `MPI_Gather`, per a poder identificar cada fitxer dins la zona de memòria habilitada en la recepció. Cal notar que `MPI_Gatherv` guardarà tots els fitxers de forma consecutiva a la memòria del procés mestre sense cap tipus de separació. Es pot veure també que la quantitat de memòria exigida per aquesta operació és perfectament assumible per al procés mestre. Tenint en compte la mida dels fitxers generats (que va des de 1 KB a 10 KB), en una execució MPI amb 1000 processos necessitaria de l'ordre de 10 MB de memòria dinàmica en el pitjor dels casos.

L'últim pas és que el procés mestre escrigui totes les dades que li han arribat al seu sistema de fitxers, al mateix lloc i mantenint els mateixos noms com si s'haguessin generat en un sistema de fitxers compartit. En aquest moment, quan Papiex acaba, la visió que l'usuari té dels fitxers generats en el sistema de fitxers local és la mateixa que tindria si fet servir l'opció estàndard.

Els desenvolupadors de Papiex estan al corrent d'aquestes ampliacions i han mostrat el seu interès per tal que siguin afegides en futures versions de l'aplicació. Actualment està pendent la seva inclusió en el repositori oficial com a branca de desenvolupament per realitzar les proves pertinents.

### 3.3.4 Instal·lació

En el moment d'instal·lar Papiex a una màquina com MareNostrum, sorgeixen alguns problemes derivats de les seves particularitats. En primer lloc, ha estat necessari utilitzar les versions en desenvolupament tant de Papiex com les seves dependències (Monitor i PAPI). Les respectives versions oficials (monitor 1.8, Papi 3.5.0 i Papiex 0.99rc9) donen certs problemes que fan que no funcioni correctament en alguns aspectes.

En aquest sentit, durant el desenvolupament del projecte s'ha mantingut contacte actiu amb els desenvolupadors d'aquestes eines per tal de trobar solucions als entrebancs que han anat sorgint. Aquesta és també la raó d'utilitzar les versions de desenvolupament, incorporant al moment les millores i sol·lucions trobades. Cal destacar que l'ajuda per part d'aquestes persones ha estat molt important per a l'èxit d'aquest projecte.

Els processadors que componen MareNostrum són IBM PowerPC970, capaços d'executar codi tant de 32 bits com de 64 bits. De fet, entre les aplicacions que s'executen diàriament es poden trobar dels dos tipus. Si es vol obtenir informació de rendiment de totes les aplicacions, és imprescindible que Papiex sigui capaç de treballar amb tot tipus de binaris.

Cal notar que per a que la interposició de crides per part de la llibreria d'instrumentació funcioni, aquesta ha de ser del mateix tipus que l'executable a analitzar. No es pot doncs treballar amb una sola instal·lació, perquè fer-ho implicaria deixar d'instrumentar executables d'un determinat tipus. La solució ha estat, com en d'altres aplicacions instal·lades al supercomputador, realitzar dues instal·lacions iguals en tot excepte en que una estarà preparada per a treballar amb 32 bits i l'altra per fer-ho amb 64.

Tenir dos instal·lacions diferents podria dificultar el procés d'extracció si s'utilitzés el llançador d'aplicacions que aquest proporciona, ja que aquest carrega les llibreries corresponents a la seva instal·lació i podrien no ser les adients pel binari a executar. Per a evitar-ho, caldria carregar l'entorn de Papiex tant de 32 com de 64 bits alhora. Fent això, quan el programa s'iniciés utilitzaria les llibreries que són compatibles, descartant les altres. S'hauria de modificar el llançador per a que pogués carregar l'entorn correcte, però no és necessari si es carrega l'entorn externament, com és el cas d'aquest projecte.

El programa està pensat per a treballar amb MPICH sense haver d'enllaçar Papiex amb la llibreria MPI. Això, que és un avantatge a MareNostrum, es torna un problema a l'hora d'utilitzar altres implementacions de MPI, com OpenMPI. I en el cas de Nord, la màquina on es realitzaran les proves (veure secció 4.1), la versió MPICH que hi ha no funciona amb el sistema de cues SLURM. Ha calgut adaptar doncs l'eina per a funcionar correctament amb OpenMPI, que sí que és capaç de treballar amb SLURM.

La instal·lació actual no inclou el suport per a mpiP degut a algunes incompatibilitats amb la versió instal·lada que no s'han arribat a solucionar. Tractant-se d'una característica afegida que no afecta directament al funcionament del sistema, la decisió ha estat deixar aquest punt per a futures ampliacions a realitzar. La informació addicional seria un bon complement a la que ja s'obté per mitja del mateix programa.

## 3.4 Integració amb el sistema

Un cop detallat el procés d'obtenció de la informació, és el moment de veure com integrar el que fins ara s'ha explicat en el sistema. Aquesta part està molt relacionada amb la de recol·lecció de la

informació (secció 3.5), pel que es faran contínues referències a ella. Al mateix temps, l'explicació de l'arquitectura quedarà dividida en aquestes dues seccions.

La integració amb el sistema consisteix en construir una infraestructura que sigui capaç d'automatitzar el procés d'extracció de la informació de rendiment de les aplicacions. Això no és una altra cosa que evitar que l'usuari hagi de dur a terme un procés manual per obtenir dades de rendiment, alhora que s'intenta també que hagi de modificar el menys possible la seva forma de treballar. Per aconseguir aquest objectiu cal treballar al nivell del sistema de cues (explicat a la secció 2.2.4). D'aquesta manera s'obté la informació de la manera més transparent possible des del punt de vista de l'usuari.

Tot seguit es detallen els requisits específics d'aquesta part, la integració al gestor de recursos SLURM i l'entorn d'enviament de treballs. S'explicarà el funcionament i el disseny del *plugin* SLURM així com la seva implementació.

### 3.4.1 Requisits principals

Com ja s'ha apuntat, el requisit principal és la transparència. L'obtenció i el tractament posterior de la informació haurà de ser totalment automàtic i sense la intervenció de l'usuari. Entrant en detall, l'usuari no haurà d'explícitament cridar a Papiex quan executa qualsevol programa en el seu treball, ni tampoc preocupar-se d'on es generen els fitxers amb aquesta informació. Aquesta tasca l'haurà de fer completament el sistema a instal·lar.

La única cosa que es pot demanar a l'usuari és que en el mateix treball s'especifiqui d'alguna manera la quantitat d'informació que s'ha d'extreure. Així, l'usuari podria decidir per un cert *job* no extreure informació (ja sigui perquè no li convé o per algun conflicte de l'extracció amb el seu programa), de la mateixa manera que podria decidir augmentar la instrumentació per obtenir més dades del seu treball.

En qualsevol cas, es definiria un nivell per defecte d'extracció per a tots els usuaris, de tal manera que en cas que no especifiquin res, s'aplicaria aquell nivell. Cal notar que això inclou tenir el sistema desactivat per defecte, i activar-lo sota demanda, cosa que aniria bé en les fases inicials de la posada en producció del sistema.

Per altra banda, el fet de modificar la configuració del sistema afegint nou codi a executar no hauria de suposar un problema, ni de funcionament ni de rendiment. Per començar, la funcionalitat i l'estabilitat del sistema no s'haurien de veure alterades: els treballs s'han de seguir executant de la mateixa forma. A més, seria desitjable que les modificacions a fer per aconseguir els objectius, tant en la configuració com en el codi, siguin mínimes i el més independents possible.

Per facilitar-ne l'administració, la seva configuració ha de ser senzilla i fàcilment modificable. També és important el fet que es pugui activar i desactivar de manera fàcil en el cas que fos necessari. Tot això es podria resumir en el requisit general de mínima interferència amb la configuració original i en les pautes de treball de l'usuari.

### 3.4.2 Anàlisi i disseny de la solució

Un bon punt de partida per a pensar el sistema és observar com treballa la versió original de PerfMiner. En aquest cas, PerfMiner es basa en la configuració dels sistemes disponibles al PDC<sup>1</sup>, on es va fer la instal·lació. Concretament, el sistema de cues utilitzat es Easy, un planificador senzill que es basa en l'activació o desactivació de la *shell* de l'usuari al fitxer de configuració */etc/passwd* per tal de donar accés o denegar-lo als nodes de càlcul.

A més, PerfMiner pretén també extreure informació de rendiment de les aplicacions que s'executen de manera interactiva (o sigui, no utilitzant el sistema de cues). Aquestes aplicacions poden ser paral·leles o no, que utilitzin paral·lelisme a nivell de memòria compartida o de memòria distribuïda. Es fa difícil doncs definir un punt d'entrada on centrar l'esforç de carregar l'entorn adient.

Tot plegat condueix a la solució d'implementar una *shell* virtual que farà d'embolcall a la *shell* original (bash) i que d'aquesta manera podrà carregar l'entorn necessari i executar Papiex. La *shell* virtual (pdcsh), serà la *shell* per defecte de tots els usuaris de la màquina per tal que es carregui també en una sessió interactiva.

Aquesta implementació xoca frontalment amb la idea exposada a l'apartat anterior. Per començar, el sistema que gestiona els recursos és totalment diferent. Al BSC-CNS s'utilitza SLURM com a gestor de recursos en els diferents supercomputadors, i Moab o Maui com a planificadors segons el cas. Aquests sistemes són molt més complexos que Easy, però alhora permeten una major flexibilitat.

A més, la política del centre és que qualsevol treball que s'hagi d'executar als supercomputadors hagi de ser enviat a les cues, per garantir un ús just i òptim dels recursos per part de tots els usuaris. Aquest fet fa que no sigui necessària l'extracció d'informació d'execucions interactives i l'atenció se centri únicament en els treballs que s'executen a través de SLURM.

Aquesta part, per tant s'ha de redissenyar completament. En primer lloc cal pensar en quin punt s'ha de canviar l'entorn de l'usuari per tal que carregui Papiex. Com que la idea és extreure informació de qualsevol aplicació que s'executi mitjançant les cues, s'ha de tenir en compte tant

---

<sup>1</sup>Center for Parallel Computers d'Estocolm

les aplicacions MPI com la resta. Per aquest motiu, interceptar "srun" per a realitzar la càrrega de l'entorn corresponent no és una opció vàlida, ja que només es podria obtenir dades d'aplicacions que s'executen en paral·lel.

Canviar la *shell* de l'usuari tampoc és la millor opció. Implicaria un canvi profund en la configuració del sistema, ja que tota l'administració es basa en que la *shell* per defecte és bash. Per altra banda, seria una possible font d'inestabilitat i d'errors, i extrauria informació també de les sessions interactives dels usuaris.

El gestor de recursos SLURM està dissenyat per a poder afegir fàcilment funcionalitats a través d'una estructura de *plugins*, o afegits. Mitjançant això, es poden implementar certes accions que es portaran a terme en determinats moments de l'execució d'un treball, i que poden ser molt diverses: control del consum, qüestions d'autenticació i seguretat, tasques de neteja, etc. Utilitzant aquesta tecnologia es pot arribar a carregar l'entorn corresponent abans d'executar el treball, i fer el que calgui al acabar amb la informació de rendiment obtinguda.

### 3.4.2.1 Els *plugins* SLURM: SPANK

En concret, s'utilitza l'anomenada arquitectura SPANK<sup>2</sup> per al desenvolupament de *plugins* per a SLURM. Consta d'una API per a llenguatge C (el llenguatge en el que SLURM està programat) amb una sèrie de funcions que podran ser implementades pel desenvolupador del *plugin*, a més d'un altre conjunt de funcions auxiliars. Les primeres, anomenades *callbacks*, seran cridades (si estan implementades) pel mateix SLURM en certs moments claus de l'execució del treball. A la figura 3.3 queda il·lustrada aquesta API, on les funcions en cursiva seran els *callbacks* que implementarà el *plugin*, i les altres les funcions auxiliars.

A cada node del supercomputador hi ha una aplicació en execució anomenada *slurmd* que és la que s'encarrega de fer la gestió de SLURM localment, gestionant les tasques que s'executen en ell i que formen part de treballs encuats al sistema. Aquesta arrencarà una altra anomenada *slurmstepd* que és la que treballarà directament amb les tasques a executar del treball. És des d'aquest procés des d'on es carregaran els *plugins*, i des d'on es realitzaran les crides *callback* per cada un d'ells.

Al fitxer de configuració de SLURM *plugstack.conf* (local a cada node), es defineixen tots els *plugins* que han de ser carregats. Per cada un d'ells, es permet especificar si son imprescindibles (en cas de no trobar-los es produiria un error), o opcionals (si no els troba no els carrega, però no produeix cap error). A més, se li poden afegir paràmetres addicionals a cada *plugin* que els arribaran en forma d'arguments del programa.

---

<sup>2</sup>SLURM Plugin Architecture for Node and job (K)ontrol

<b>spank</b>
# spank_options[] : spank_option
+ <i>init</i> ()
+ <i>fini</i> ()
+ <i>slurm_spank_init</i> ()
+ <i>slurm_spank_local_user_init</i> ()
+ <i>slurm_spank_user_init</i> ()
+ <i>slurm_spank_task_init</i> ()
+ <i>slurm_spank_task_post_fork</i> ()
+ <i>slurm_spank_task_exit</i> ()
+ <i>slurm_spank_exit</i> ()
+ <i>spank_symbol_supported</i> ()
+ <i>spank_remote</i> ()
+ <i>spank_get_item</i> ()
+ <i>spank_getenv</i> ()
+ <i>spank_setenv</i> ()
+ <i>spank_unsetenv</i> ()

Figura 3.3: Diagrama de la API SPANK

Els *plugins* SPANK poden ser carregats en dos contextos diferents: local i remot. En remot és *slurmd/slurmstepd* qui carrega el *plugin* quan el treball comença. En context local, en canvi, és un *srund*, dins del mateix treball, qui carrega el *plugin* (una execució paral·lela dins del treball). Això significa que dins d'un mateix treball el *plugin* es pot haver carregat més d'un cop en contextos diferents, pel que SPANK ofereix la funció *spank\_remote* per a que dins del *plugin* es pugui determinar en quin àmbit s'ha carregat, ja que les tasques a fer poden ser diferents.

La seqüència de crides en context remot es mostra a la figura 3.4, i va des de l'inici del treball fins a la seva finalització. Cal notar que la seqüència mostrada conté totes les crides possibles, però el *plugin* pot implementar només un cert nombre. El procediment seguit queda descrit a continuació. Només arrencar el treball, i encara amb privilegis, el *plugin* es carrega i s'executa la funció *init*. Posteriorment, es revoquen els privilegis i es comença a executar com a l'usuari que ha enviat el treball i amb els seu entorn.

En aquest punt és on es crida a *user\_init*, la primera funció dins el *plugin* amb l'entorn propi del treball carregat i que s'executa abans de començar amb les diferents tasques. Per a cada una d'elles, SLURM haurà d'executar el *fork* corresponent per crear el procés de la tasca. Entre aquest moment i el moment de carregar la tasca (*execve*) es realitza una crida a *task\_init*. Cal notar que en aquest punt encara no s'ha començat a executar la tasca, perquè ni tan sols està carregada la seva imatge al procés fill.

Just abans de començar l'execució normal de la tasca, es fa una última crida a *task\_post\_fork* per possibles ajustaments abans que el procés comenci a caminar. A partir d'aquí, *slurmstepd* ha

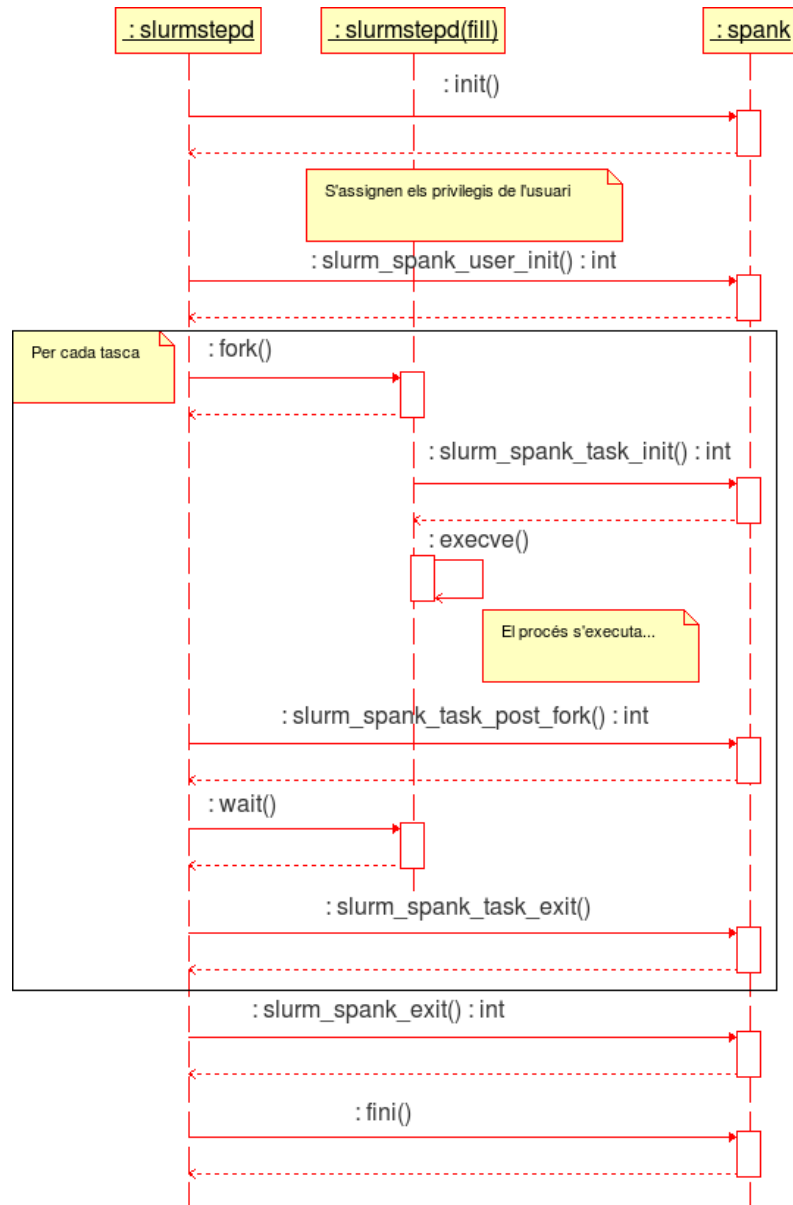


Figura 3.4: Diagrama de seqüència dels *plugins* SPANK

d'esperar (amb la crida a sistema `wait`) la finalització de la tasca. Un cop finalitzada, es realitzarà la crida a `task_exit` per realitzar accions en el moment d'acabar la tasca.

Cal diferenciar aquesta última funció amb la de `exit`, ja que aquesta es realitza quan el treball acaba, mentre que `task_exit` s'executa cada cop que acaba una tasca. A la funció `exit` encara es conserva l'entorn de treball propi per si cal fer-ne ús. Per últim, la funció `fini` és executada amb el treball ja finalitzat, i amb privilegis altre cop de l'usuari SLURM. Després d'això, el *plugin* acaba la seva execució.

En el context local, dins d'una execució en marxa de `srn`, només la funció `local_user_init` és cridada (a part de les implícites `init` i `fini`). El moment de l'execució és el mateix que `user_init` en el context remot, és a dir, abans que `srn` arrenqui les tasques corresponents.

### 3.4.2.2 Disseny del *plugin*: `spank_perfminer`

Com s'ha pogut observar, aquesta tecnologia permet una integració molt neta en el sistema, ja que s'aprofita una característica del mateix gestor de recursos per implementar una nova funcionalitat o servei, sense haver de fer grans canvis en l'arquitectura o la configuració del conjunt. A l'hora de dissenyar el *plugin*, diverses decisions han hagut de ser preses per tal d'arribar la implementació final, tenint en compte les funcionalitats que ha de complir:

- Càrrega de l'entorn de Papiex
- Gestió dels directoris temporals i fitxers d'informació generats
- Enviament de les dades obtingudes
- Possible neteja dels fitxers generats

Algunes d'aquestes característiques, com l'enviament de les dades, queda a cavall entre aquesta secció i la secció de recollida de la informació (3.5), pel que la seva justificació quedarà ajornada per ser tractada en aquell punt. A la figura 3.5 es pot observar el disseny UML del *plugin* implementat, mentre que a les figures 3.6 i 3.7 es troben els diagrames de seqüència que descriuen el funcionament del *plugin*. Tot seguit es detalla aquest funcionament raonant sobre les possibles solucions per a cada punt.

La funció principal del *plugin* és carregar l'entorn de Papiex en un cert moment per tal que totes les tasques del treball siguin tractades. Cal però decidir quin és el moment més adient per realitzar aquesta càrrega. Tenint en ment la seqüència mostrada a la figura 3.4, sorgeixen diverses

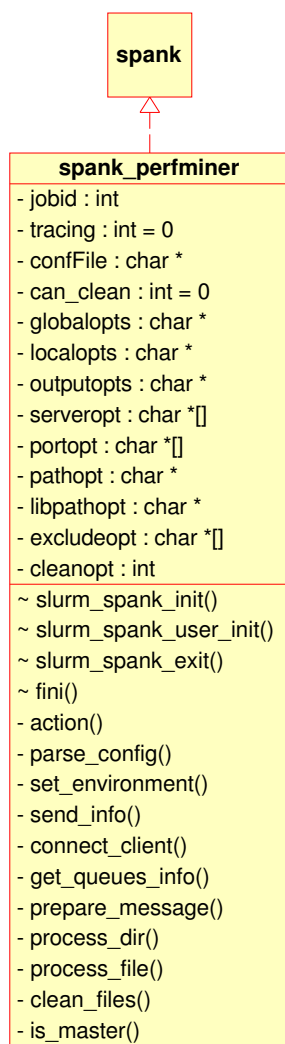


Figura 3.5: Diagrama de classes del *plugin* spank\_perfminer

possibilitats, dins de les funcions cridades en moments d'inicialització. A priori, les funcions candidates per realitzar aquesta feina serien `init()`, `slurm_spank_user_init()`, `slurm_spank_task_init()` i `slurm_spank_post_fork()`.

La primera opció queda descartada degut a que encara no es té accés a l'entorn del treball (es treballa amb privilegis), mentre que la última, havent-se ja creat el procés fill, impossibilita canviar-li l'entorn d'execució. Queden per tant `user_init` i `task_init` com a possibles solucions. Les dues són igualment vàlides, ja que en el moment de la crida ja tenen accés a l'entorn del treball, i la modificació es produiria abans d'executar la tasca o tasques del treball. Tot i això, `user_init` es perfila com a millor solució, ja que només s'executa un cop (no cal més per a carregar l'entorn). `Task_init`, per la seva banda, carregaria l'entorn cada cop que s'anés a executar una nova tasca, perdent eficiència.

Per facilitar la portabilitat d'aquest afegit així com la seva administració, sorgeix la necessitat de separar la part funcional del programa de la seva configuració. D'aquesta manera, el *plugin* es podrà instal·lar fàcilment a diversos supercomputadors sense haver de fer grans canvis. Totes les opcions del programa quedaran definides dins d'un fitxer de configuració (*perfminer.conf*). Entre les opcions es troben l'entorn que caldrà carregar, la definició dels diversos nivells d'extracció i d'altres que es detallaran més endavant.

Aquest fitxer es llegiria a l'inici del *plugin*, a la funció *slurm\_spank\_init*. Es convertiria d'aquesta manera en la primera tasca que fa el *plugin* després de carregar-se. De nou, amb la portabilitat al cap, el programa no força que el fitxer de configuració estigui en una ubicació concreta, sinó que la rebrà com a argument al arrencar (essent un paràmetre addicional al fitxer de configuració de SLURM *plugstack.conf*).

D'entre l'entorn que ha de carregar el *plugin* figura el lloc on es generaran els fitxers amb la informació de rendiment. De la mateixa manera que la resta de l'entorn, aquest paràmetre és definit al fitxer de configuració i pot apuntar a un directori dins de GPFS o del disc dur local de cada node (*scratch*). D'una o altra manera, s'haurà de crear el directori en cas que no hi sigui per evitar errors en temps d'execució, així com oferir la possibilitat d'eliminar els fitxers generats.

En aquest punt cal destacar l'estructura de directoris escollida. Dins del directori definit com a temporal per escriure-hi els resultats obtinguts, cada treball crearà un subdirectori únic per a ell mateix i és allà on emmagatzemarà les dades generades. D'aquesta manera, per un costat totes les dades estan a la mateixa ubicació, però alhora classificades per treball. Aquesta estratègia facilita tant la identificació i recuperació de la informació com la seva neteja, ja que cada fitxer amb informació queda aïllat amb la resta del seu mateix treball sense interferir en la resta.

Com a complement, aquesta solució aporta un grau afegit d'eficiència en cas que s'utilitzi GPFS com a suport d'emmagatzematge. En aquesta situació, cada treball només escriurà els fitxers que calgui sobre el seu subdirectori, disminuint la sobrecàrrega d'escriptures sobre el mateix directori. Si es generessin tots els fitxers de tots els treballs dins el mateix directori, l'eficiència i l'estabilitat del sistema es veurien compromeses.

Quan el treball finalitza, quedaran totes les dades dins del directori temporal mencionat a l'espera de ser tractades. El *plugin* s'haurà d'encarregar de fer-les arribar al destí, un servidor intermedi entre la base de dades i el supercomputador. Cal recordar que des dels nodes de computació no es té accés a l'exterior, només al conjunt de nodes del supercomputador a més d'algun servidor auxiliar.

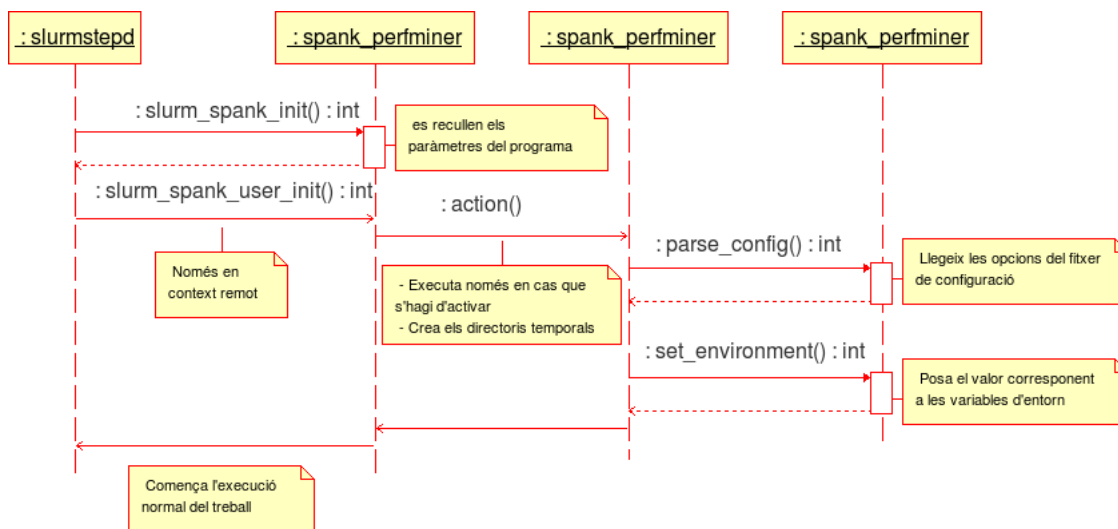


Figura 3.6: Diagrama de seqüència de la inicialització al *plugin* `slurm_spank_perfminer`

Altres cops, com al moment de carregar l'entorn, són diverses les funcions candidates per implementar aquesta tasca: `slurm_spank_task_exit`, `slurm_spank_exit` i `fini`. Per evitar interferir en el desenvolupament normal del treball, el tractament de la informació s'hauria de dur a terme un cop s'ha finalitzat correctament en la seva totalitat. Per això, `task_exit` no és una opció viable, ja que es cridaria al finalitzar cada tasca del treball. Tampoc és una bona alternativa utilitzar `fini`, ja que en aquell moment s'ha perdut constància de l'entorn del treball.

Queda, per tant, la funció `slurm_spank_exit` com a candidata per realitzar l'enviament, ja que s'executa havent acabat el treball però encara amb l'entorn correcte per tal de recuperar les dades. Acabat el *job*, tota la informació estarà dins del subdirectori corresponent dins el directori temporal, pel que serà fàcil recollir-la i enviar-la. El *plugin* es connectarà amb el servidor de recollida a través de la xarxa i li enviarà les dades obtingudes.

En principi, per assegurar la disponibilitat, si no es pot establir la connexió amb un servidor s'intentarà connectar amb un altre. Així, es poden definir 5 servidors de recollida dins del fitxer de configuració als quals connectar-s'hi. Si es donés el cas, es podria aplicar un algorisme de balanceig de càrrega entre els diferents servidors, per tal que cada cop que es realitza un enviament s'escollís un servidor a l'atzar d'entre els que són disponibles. Lògicament, no és necessari definir tots els servidors. El sistema pot funcionar correctament amb un servidor.

De tota manera, cal pensar què fer si no es pot contactar amb cap servidor de recollida. Les opcions possibles serien reintentar la transmissió durant un temps determinat o no enviar-les i alliberar els recursos per un altre treball que hi hagi a la cua. Com que un dels requeriments marcats és que s'ha d'interferir el menys possible, no es produirà l'enviament de les dades si no es troba cap servidor.

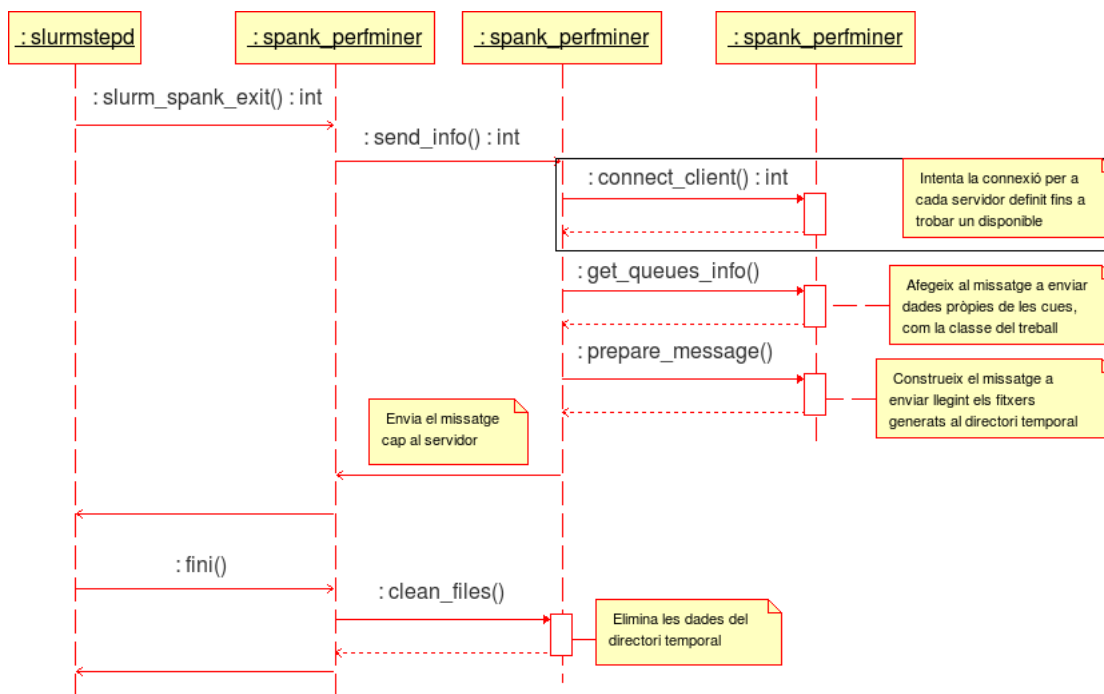


Figura 3.7: Diagrama de seqüència de la finalització al *plugin* *spank\_perfminer*

En aquest supòsit, les dades generades es perdrien ja que no hi ha cap sistema per recuperar-les un cop el *plugin* ha acabat la seva feina amb el treball.

Posteriorment, havent enviat o no les dades, s'hauria de procedir a la neteja dels fitxers temporals generats per Papiex. El *plugin* eliminarà tots aquells fitxers generats en aquell treball (tot el que hi ha sota el subdirectori creat per a l'ocasió), ja que no seran necessaris més endavant. Aquesta neteja és especialment important en el cas d'utilitzar el disc local com a temporal, ja que si no es netegés s'aniria omplint indefinidament amb els fitxers temporals fins arribar al punt de saturar l'espai disponible.

De tota manera, es pot escollir no esborrar les dades generades per a una possible depuració del funcionament del programa, així com establir si s'han d'eliminar els fitxers d'un sistema de fitxers compartit (GPFS) o no compartit (*scratch*). En el cas de GPFS, només un dels nodes ha de realitzar la neteja (ja que tots accedeixen al mateix sistema de fitxers), mentre que sobre disc local cada node ha de fer la seva neteja particular.

La neteja s'ha de dur a terme necessàriament després d'haver intentat enviar les dades. Els dos moments possibles són la mateixa funció `slurm_spank_exit`, on es realitza l'enviament, i la funció `fini`. Les dues són igualment vàlides, sempre que en l'últim cas es tingui disponible la ruta del directori que s'ha d'esborrar (cal recordar que a `fini` no es conserva l'entorn del treball).

### 3.4.2.3 Nivells d'extracció

Un altre aspecte a considerar és la flexibilitat de la solució original de PerfMiner. La informació a extreure seria sempre la mateixa, i s'extrauria sempre per a tots els treballs. Seria millor poder escollir per a cada treball si cal extreure o no informació, i quina s'ha d'extreure. Amb aquesta finalitat, el sistema està pensat per a definir diversos nivells d'extracció d'informació, de manera que a cada nivell s'obtenen unes determinades dades.

Utilitzant els nivells, es deixa total llibertat a l'administrador per definir tants com vulgui i amb el conjunt de dades que cregui convenient per a cadascun. Es poden implementar diverses variants amb aquesta filosofia, com per exemple una definició de nivells jeràrquica, on cada nivell que s'avança s'obté més informació, que també pot implicar més sobrecàrrega sobre el programa analitzat.

Una alternativa seria definir nivells no jeràrquics, on els nivells inferiors no estiguessin inclosos en els superiors. D'aquesta forma, es podria arribar a extreure conjunts de dades totalment heterogenis, podent escollir després el que més interessi en cada cas.

Si no es desitja treure informació, hi haurà reservat el nivell 0, que farà que no es produeixi l'execució de Papiex i per tant no s'obtinguin dades d'aquell treball. Aquesta possibilitat resulta útil en cas de trobar algun problema de compatibilitat entre el programa a analitzar i el sistema, o també quan es vol executar proves de rendiment que necessiten les menys interferències possibles. Un altre situació on seria recomanable desactivar aquest sistema seria si es volen utilitzar altres eines d'obtenció de dades de rendiment, com les descrites a la secció 2.4.

Els nivells quedaran definits al fitxer de configuració del *plugin*, i aniran numerats seqüencialment a partir del 0. El sistema tindrà un nivell d'extracció per defecte, quedant a mercè dels administradors decidir quin nivell establir com a estàndard. Per exemple, podria ser interessant definir el 0 (no extracció) en les primeres etapes d'implantació, de manera que només es treballaria amb aquells usuaris que fessin explícit quin nivell extreure. En cas de detectar problemes, seria més fàcil acotar-los i solucionar-los.

L'usuari ha de poder especificar d'alguna manera en els seus treballs quin nivell vol utilitzar. La solució ha de ser fàcilment integrable dins l'entorn d'encuament de treballs, així com també dins el mateix treball. A més cal tenir en compte que els treballs no s'executen en el moment de ser enviats, sinó en el moment que el planificador ho decideix.

L'opció més viable és utilitzar el mateix *script* del treball com a via d'interacció amb el sistema, mitjançant una nova directiva anomenada *mining\_level*, que prendrà el valor del nivell desitjat.

Aquesta directiva s'afegeix a les ja existents al treball i amb el mateix format, com les del nombre de processadors demanats o el temps de duració (veure secció 2.2.4, figura 2.4).

La directiva especificada es convertirà en una variable d'entorn que s'afegirà al mateix treball, de tal forma que quan s'executi i el *plugin* sigui carregat, pugui esbrinar fàcilment quin és el nivell d'extracció que s'ha d'aplicar per aquell treball en concret. Aquesta solució és molt neta, ja que només afecta al treball en qüestió (cada treball podrà tenir un nivell diferent). A més, l'impacte en el sistema original es pot considerar mínim, i els treballs que l'usuari tenia segueixen sent vàlids.

En el cas que no s'especifiqui res, el sistema haurà d'assumir un comportament per defecte, que també quedarà definit al fitxer de configuració. D'aquesta manera, es pot canviar de manera fàcil el comportament que té el sistema en el cas dels treballs que no tinguin la nova directiva, i fixar-lo en el nivell 0 (no extracció), o en qualsevol altre.

### 3.4.3 Implementació i instal·lació

La implementació d'aquesta part del projecte es divideix en dues parts. Per una banda, el gruix del treball es concentra en la creació del *plugin* de SLURM, i per l'altra hi ha la modificació de la comanda d'enviament de treballs per a que reconegui la nova directiva. A continuació queden detallats aquells aspectes més rellevants de la implementació de cadascuna d'elles.

En el desenvolupament del *plugin* de SLURM s'ha d'utilitzar el llenguatge C, ja que la API que proporciona SLURM, i en concret SPANK és en aquest llenguatge. El principal avantatge és la seva eficiència, ja que és un programa que es carregarà cada cop que es posi en marxa un treball, i ha de ser el més ràpid possible per no destorbar al mateix treball. Per contra, algunes tasques com la gestió de cadenes de caràcters (*strings*), o la gestió de la memòria esdevenen molt més complexes per la falta de suport del mateix llenguatge.

Per suplir part d'aquestes carències, s'han utilitzat una sèrie de llibreries auxiliars anomenades *etools*, que faciliten feines molt habituals com la lectura de fitxers o necessitat d'estructures de dades com una taula *hash*. Aquestes llibreries són de codi obert, i es compta amb el coneixement i l'aprovació del seu desenvolupador per a ser utilitzades en aquest projecte.

La totalitat del codi d'aquest afegit està continguda en un mateix fitxer font *.c*, que segons les especificacions dels *plugins* haurà de ser compilat com a llibreria dinàmica de Linux (*.so*). D'aquesta manera, SLURM podrà carregar en temps d'execució aquesta llibreria i cridar les funcions *callback* que tingui definides. Paral·lelament, es troba el fitxer de configuració del *plugin*, que vindrà amb les opcions més habituals i comentat per a que sigui més fàcil ajustar els diversos paràmetres.

Per a l'ocasió, es crearà la següent estructura de directoris per allotjar el *plugin* en la seva instal·lació:

- *etc/* : Aquí es guardarà el fitxer de configuració del *plugin* (*perfminer.conf*).
- *lib/* : Aquest directori contindrà el *plugin* en si mateix (*spank\_perfminer.so*).

S'ha decidit, a més, utilitzar */gpfs/apps/PERFMINER* per a contenir els directoris mencionats, de manera que queda aïllat de tota la configuració pròpia de SLURM o Moab per tal d'interferir el menys possible en la gestió habitual de la configuració d'aquests elements tan importants per al funcionament del supercomputador.

La instal·lació seguirà la mateixa política que el programari similar als supercomputadors del centre, que serà mitjançant paquets rpm. Gràcies a això és fàcil instal·lar o desinstal·lar el programa i tenir un control més ajustat sobre les versions instal·lades. Com la resta de components del sistema de cues, aquest *plugin* s'ha integrat en un repositori CVS, unificant l'administració d'aquest aspecte de la màquina. Amb aquesta solució es pot instal·lar una versió anterior plenament funcional en segons, en cas que es detecti algun problema en la versió que hi hagi instal·lada.

Parlant ja de la part de l'usuari, s'ha modificat la comanda *mnsuubmit* per a que reconeguéss la nova directiva. Aquest programa llegeix el *script* que l'usuari ha enviat per executar en busca dels paràmetres que van precedits amb els símbols "# @". La nova directiva és un d'aquests, pel que només cal afegir una condició per a l'ocasió. El que farà el programa en cas de trobar la directiva és afegir en l'entorn del treball la variable d'entorn "PERFMINER" amb el valor corresponent al nivell d'extracció especificat.

Els problema més important en aquesta part del projecte ha estat sens dubte obtenir una plataforma on realitzar les proves de funcionament. Per a provar la funcionalitat bàsica, n'hi ha hagut prou amb una instal·lació de SLURM en un portàtil. Tot i ser útil al principi, hi ha certes parts del programa que estan subjectes al paral·lelisme (diversos nodes). Ha calgut utilitzar un petit clúster de proves (veure secció 4.1), amb dos nodes, per a provar aspectes com la resposta davant de treballs paral·lels, o l'actuació sobre els fitxers temporals en els diferents discos locals.

Cal remarcar que a cada entorn de proves que s'ha fet servir s'ha hagut d'instal·lar i provar eines necessàries, com el sistema de cues, les eines d'extracció o les llibreries MPI, així com provar el seu correcte funcionament abans d'integrar la nova eina.

## **3.5 Recol·lecció de la informació**

Després de veure com s'extreu la informació de les aplicacions i com s'integra aquest mecanisme amb el sistema existent, cal veure com es farà la recollida de totes aquestes dades i el procés corresponent per a poder emmagatzemar-les en una base de dades. Anàlogament a com s'ha tractat l'apartat anterior, el primer serà descriure els requisits més destacats que té aquesta part del sistema, per després realitzar l'anàlisi i disseny de la solució. Per últim, s'acabarà amb una explicació de la implementació i la instal·lació dels servidors de recollida de les dades.

### **3.5.1 Requisits principals**

La recollida de la informació generada per totes les aplicacions d'un supercomputador no és una tasca trivial, ja que depèn tant de la mida del supercomputador, com del nombre d'aplicacions que s'executen. Com que el sistema a desenvolupar haurà de treballar en un supercomputador de gran envergadura, s'ha de plantejar un sistema que sigui escalable a les circumstàncies d'aquest supercomputador i de l'ús que se'n fa. S'haurà de tenir en ment l'escalabilitat doncs a l'hora de fer el disseny per a que la recollida es pugui fer en condicions adients.

Per la mateixa raó, el temps de resposta i la velocitat de procés prenen una importància cabdal en aquesta part. Un supercomputador en plena producció executarà en cada instant centenars de treballs, dels que es podrà estar obtenint informació de rendiment. A més, cal tenir en compte que al no aturar-se el flux d'informació que s'anirà generant serà també considerable. Per això cal un sistema capaç d'absorbir tot aquest flux de dades a un ritme prou ràpid com per no arribar a saturar.

Com que aquest sistema de recollida s'haurà d'integrar en la infraestructura existent, haurà de tenir el menor impacte possible en el rendiment original. El supercomputador disposa de múltiples serveis que utilitzen els serveis que aquest procés de recollida també hauria de compartir. Això vol dir que haurà de ser lleuger i consumir els mínims recursos possibles tant de procés, memòria, disc o xarxa.

### **3.5.2 Anàlisi i disseny de la solució**

Tot seguit es desglossa l'anàlisi i el disseny de la solució proposada per aquesta part del sistema. Es tractarà en primer terme l'estratègia de recollida escollida, per continuar tot seguit amb el detall de l'arquitectura escollida per a aquest procés i la descripció del programari que hi intervé.

### 3.5.2.1 Estratègia de recollida

La solució original de PerfMiner es va dissenyar per a poder funcionar correctament en supercomputadors de l'ordre de centenars de processadors. En canvi, per a que la recollida funcioni eficientment sobre un supercomputador de l'ordre de milers de processadors cal analitzar bé el procediment escollit, ja que el flux d'informació que es va generant contínuament és molt més gran, i la complexitat del sistema pot invalidar el plantejament inicial.

PerfMiner planteja la recollida de manera passiva. Quan un treball acaba, simplement deixa les dades de rendiment obtingudes en un directori temporal, i serà un altre procés el que s'encarregui d'empaquetar-les i anar-les a buscar posteriorment. Dit d'una altra manera, la recollida s'activa des de fora del supercomputador. La decisió de què i quan es recull es pren externament, per exemple planificant-la cada cert temps.

Aquesta proposta, però, té certs inconvenients a l'hora de fer-la encaixar en un sistema més gran. En primer lloc, per a que aquesta solució sigui factible, el sistema de fitxers on s'allotgen els fitxers temporals ha de ser accessible des del servidor des d'on es fa la recollida. Altrament, s'hauria d'utilitzar un dels nodes d'accés del supercomputador com a enllaç. A més, en el cas que s'utilitzés els discos locals de cada node com a emmagatzemament temporal, també s'hauria d'accedir a cada node de càlcul on hi han dades per a recollir-les.

La gestió doncs es complica, ja que en l'últim cas el sistema recol·lector hauria de saber a quins nodes accedir per recuperar informació, o realitzar una enquesta node a node per veure si té alguna cosa a recollir. En un o altre cas, s'introdueix una certa sobrecàrrega innecessària per a localitzar les dades temporals.

Per això, la decisió en aquesta part ha estat la de dissenyar completament una solució que s'adeqüi millor a la escala de treball del sistema. La recollida es planteja doncs a la inversa: seran els mateixos nodes de càlcul els que, un cop acabat el treball, facin arribar el que han obtingut cap al servidor. Ara, la recollida és des de dins cap a l'exterior.

Amb aquesta nova estratègia, es solucionen els inconvenients esmentats. Per un costat, el servidor de recollida ja no ha de tenir accessible el sistema de fitxers, sinó que n'hi ha prou amb que tingui connexió amb els nodes de càlcul del supercomputador. Com a conseqüència, ja no importa quina de les dues solucions d'emmagatzemament temporal (GPFS o disc local) es faci servir, ja que des de l'origen (un node de càlcul), qualsevol dels dos sistemes de fitxers és visible.

Per altra banda, com que la interacció comença des del node de càlcul, el servidor no ha de conèixer qui li ha d'enviar informació, sinó simplement estar disponible per a rebre el que li arribi des dels

nodes de càlcul. No caldrà doncs tenir un registre dels treballs acabats i a on s'executaven, ni tampoc realitzar connexions innecessàries. Amb tot, el control i la gestió de la informació utilitzant aquest mètode és molt més senzill i escalable.

Tenint l'escalabilitat i la disponibilitat en ment, aquesta proposta també permet que hi hagin diversos servidors de recollida, tal com s'avançava en el disseny del *plugin* SLURM a la secció 3.4.2.2. Aquests servidors seran coneguts per tots els nodes de càlcul, de manera que és fàcil repartir la càrrega entre ells i assegurar una alta disponibilitat en cas de que algun d'ells falli.

Una possible alternativa a la solució proposada per aquest problema era aprofitar la infraestructura existent a MareNostrum per a la monitorització. Concretament, per a aquesta finalitat s'utilitza GGCollector[Art06, Art07] treballant sobre Ganglia[Gan]. De manera resumida, aquests sistemes ofereixen una recollida escalable de certa informació a nivell de tot el clúster, de forma que es pugui monitoritzar l'estat de la màquina de manera global segons uns paràmetres anomenats mètriques.

Aquesta solució era, a priori, la més adient ja que permetria utilitzar un sistema prou desenvolupat, provat i desplegat dins del supercomputador. A més, les modificacions dins del sistema haurien estat mínimes, i es tindria la garantia de treballar sobre un sistema completament escalable. Tot i això, certs obstacles van fer decantar la balança cap a la solució que finalment s'ha portat a terme, i que tenen a veure amb el funcionament intrínsec d'aquest conjunt d'aplicatius.

El primer problema que sorgeix és que GGCollector està orientat al node, i no al treball. Com és obvi, en un sistema de monitorització com aquest cal obtenir la informació requerida per a cada node per després poder mostrar-la globalment. De cada node n'obté una sèrie de mètriques i les emmagatzemarà organitzades segons el node i la mètrica. La necessitat, però, és una altra. Un treball podria estar en execució en diversos (fins i tot centenars) de nodes diferents. Per això caldria tenir una recollida que girés entorn del concepte de treball.

A més, GGCollector està pensat per a mantenir un consum acotat de memòria, pel que només reserva un espai per a una certa mètrica i node. Això faria difícil tant relacionar cada mètrica amb el treball i fins i tot procés o *thread* al que pertany. Addicionalment, si dos treballs diferents estiguessin compartint el node, no es podria guardar la informació de tots dos alhora per a la mateixa mètrica.

L'única solució passaria per definir mètriques compostes, on en la mateixa mètrica ja es tingués en compte quin treball, procés i *thread* l'ha generada. De nou apareix un nou contratemps: GGCollector requereix que totes les mètriques que obtingui estiguin definides prèviament, i per tant no seria possible generar-les dinàmicament per a cada nou treball que entrés a execució. Encara que

això fos possible, la quantitat de mètriques afegida seria insostenible i repercutiria seriosament en el seu funcionament normal.

Per dimensionar numèricament la càrrega que l'estratègia escollida suposaria per al sistema, s'ha realitzat un estudi dels treballs executats durant una setmana a MareNostrum. D'aquesta manera es pot arribar a obtenir una estimació de la quantitat d'informació que s'hauria de recollir i amb quina freqüència. D'aquests treballs ens interessa el moment en que acaben, ja que és el moment en el que es produirà l'enviament de les dades, així com el nombre de processadors que tenia assignats, del que es pot derivar la quantitat d'informació que s'enviarà.

La mida dels fitxers generats amb les mesures de rendiment és de l'ordre de 1 a 10 KB (veure secció 3.3.1). Sabent que Papiex genera un fitxer per a cada procés o *thread*, es prendran una sèrie de consideracions abans de continuar. Primer, s'entén que a cada processador demanat en el treball només s'hi executarà un procés o un *thread*. Per tant, el nombre de fitxers generat es pot considerar proporcional al nombre de processadors.

En segon lloc, cal notar que en un mateix treball es poden realitzar diverses execucions paral·leles (que multiplicarien el nombre de fitxers generats) i algunes execucions seqüencials. No obstant, a efectes pràctics d'aquests càlcul, es prendrà com a treball tipus un que només contingui una execució paral·lela amb tants processadors com hagi demanat. La raó és senzilla: a més de simplificar l'anàlisi, aquest model és el que més s'aproxima a la majoria de treballs que s'executen a MareNostrum. Pocs treballs duen a terme més d'una execució paral·lela dins el mateix, i en el cas que sigui una execució gran, si hi hagués algun procés seqüencial addicional al mateix treball seria menyspreable.

En tercer lloc, de cara a compensar d'alguna manera aquestes aproximacions i obtenir números més prudents, assumirem que tots els fitxers generats seran de la mateixa mida, que serà de 10 KB. Tot i que en molts casos la mida és menor, cal deixar marge de maniobra per si s'afegeixen més mètriques i la mida mitjana del fitxer augmenta.

En quart lloc, s'assumeix que es pot treballar a la màxima velocitat de transmissió que la xarxa ofereix, sense tenir en compte interferències d'altres transferències que es puguin estar duent a terme al mateix temps. Tampoc es té en compte la latència o el temps de propagació de la informació, que es considera menyspreable comparat amb els temps de transferència que es tractaran.

Per últim, tampoc es té en compte la sobrecàrrega introduïda pels diferents protocols de xarxa utilitzats. Això inclou les diverses capçaleres introduïdes, i el cost d'establir les connexions pertinents.

Nombre de Processadors	Quantitat de dades a transferir	Temps de transferència
1	10 KB	81.9 $\mu$ s
2	20 KB	163.8 $\mu$ s
4	40 KB	327.7 $\mu$ s
8	80 KB	655.3 $\mu$ s
16	160 KB	1.31 ms
32	320 KB	2.6 ms
64	640 KB	5.2 ms
128	1.25 MB	10.5 ms
256	2.5 MB	21 ms
512	5 MB	41.9 ms
1024	10 MB	83.9 ms
2048	20 MB	167.8 ms
4096	40 MB	335.6 ms

Taula 3.1: Temps teòrics de transferència i mides aproximades dels fitxers temporals

En el cas més senzill, on hi ha un treball de només un processador que executa una aplicació seqüencial, es generaria un fitxer de sortida amb les dades de rendiment. S'haurien d'enviar doncs 10 KB a través de la xarxa. Com que la xarxa és una Gigabit Ethernet, en teoria l'enviament es realitzaria en aproximadament 16 microsegons:

$$T_{trans} = 10 \text{ KBytes} \cdot \frac{1024 \cdot 8 \text{ bits}}{1 \text{ KByte}} \cdot \frac{1 \text{ s}}{1 \cdot 10^9 \text{ bits}} = 81.9 \mu\text{s}$$

Fàcilment, es pot ampliar aquest resultat a mides de treballs més habituals, com s'observa a la taula 3.1. Es pot veure que en el cas d'un treball de gran mida, com són els de 4096 processadors, la quantitat de dades a enviar no supera els 40 MB d'informació, pel que l'enviament es pot fer en l'ordre de mil·lsegons. És fàcil veure també que en el cas pitjor (si acabessin tots els treballs que hi han al sistema simultàniament), la quantitat d'informació que es mouria per la xarxa no superaria els 100 MB. Aquesta quantitat d'informació és perfectament assumible per part de la xarxa existent, ja que a través d'aquesta es munta el sistema de fitxers que mou dades molt més grans, i no suposaria una gran interferència.

En la setmana analitzada, van acabar 17125 treballs, cosa que implica que la mitjana de treballs acabats per minut està al voltant de 1,7 *jobs*/min. Dit d'una altra manera, cada 36 segons acaba un treball (de mitjana). Lògicament el comportament del sistema no és ni molt menys constant, cosa que fa que durant una estona no n'acabi cap, mentre que més tard n'acabin molts de pràcticament seguits, en molts casos de treballs de pocs processadors.

De fet, si s'analitzen els espais de temps (*slots*) entre els que no es produiria cap transferència, es descobreix que en el 19% dels casos aquest interval no supera un segon de durada (figura 3.8,

taula 3.2), tenint a més el 8% del total de transferències que es realitzarien al mateix segon. De tota manera, queda clar que bona part de les transferències (un 31%) anirien separades de més de 25 segons.

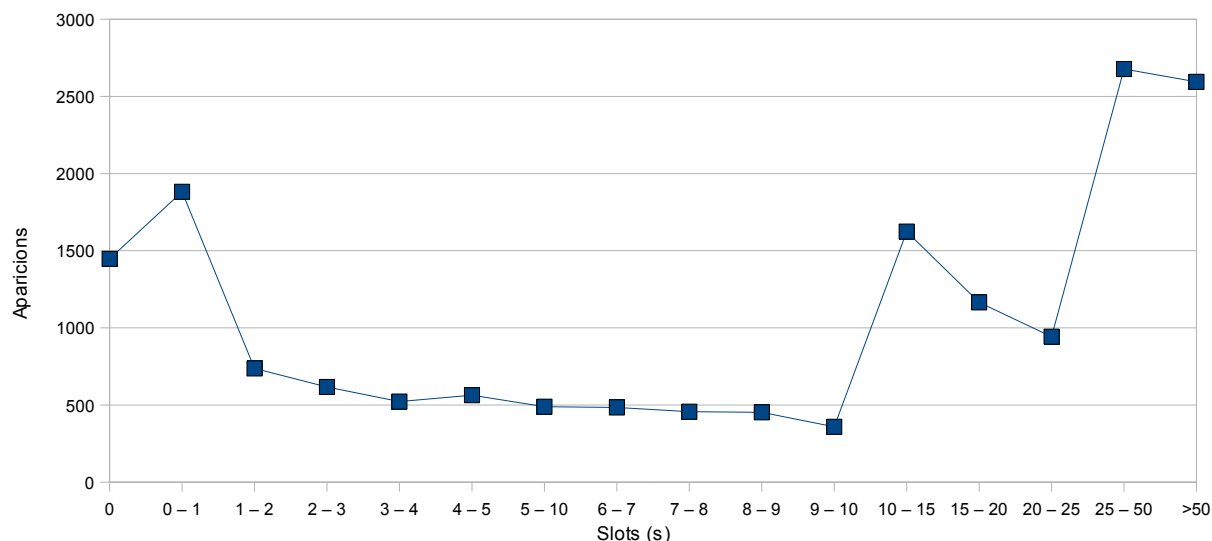


Figura 3.8: Distribució dels *slots* temporals entre transferències

Slots	Aparicions	Percentatge
0	1447	0.08
0 - 1	1882	0.11
1 - 2	737	0.04
2 - 3	617	0.04
3 - 4	523	0.03
4 - 5	564	0.03
5 - 10	2243	0.13
10 - 15	1624	0.09
15 - 20	1165	0.07
20 - 25	943	0.06
26 - 50	2677	0.16
>50	2594	0.15

Taula 3.2: Distribució dels *slots* temporals i el percentatge del total de les transferències

Per altra banda, es pot analitzar la distribució d'aquests *slots* temporals relacionada amb la mida de treball que hauria acabat, que és proporcional a la quantitat d'informació a recollir. A la figura 3.9 es pot veure aquesta distribució, que ve derivada de l'anterior. Queda reflectit que els treballs que podrien donar més problemes (els que només deixen menys de 5 segons entre arribada i arribada), són precisament els treballs més petits. Només cal veure que el pic més gran de intervals de

menys de 1 segon està format per treballs de 4 processadors. En canvi els treballs grans, tot i que suposarien un pic de feina per al servei de recollida en el moment d'acabar, no tindrien un impacte important en el funcionament del sistema.

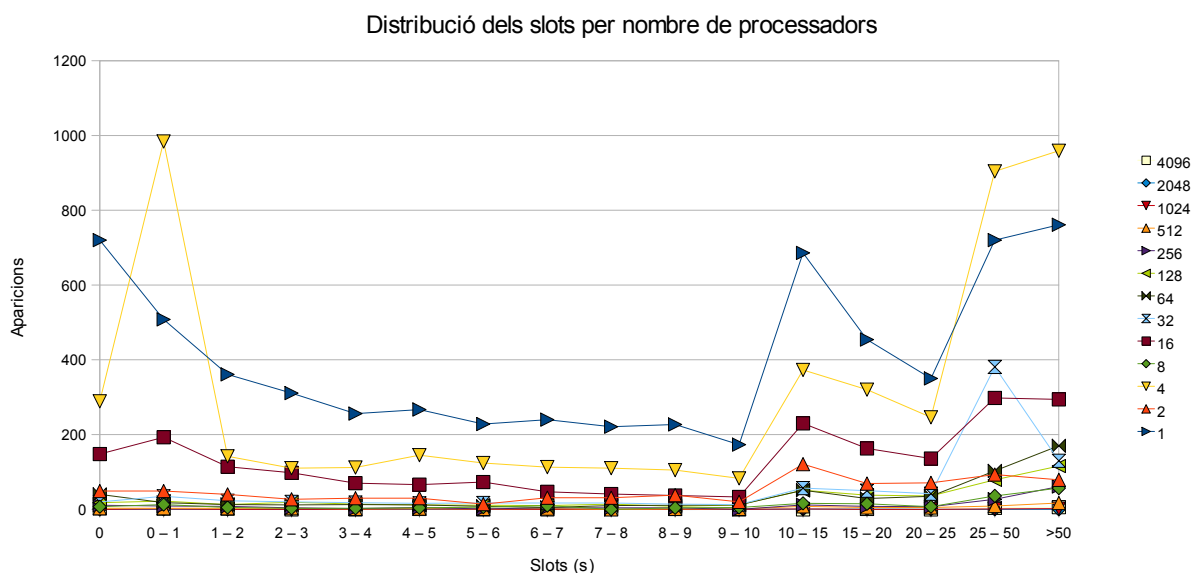


Figura 3.9: Distribució dels *slots* temporals entre transferències per mida de transferència

Amb aquests resultats a la mà, es conclou que tot i tenir ràfegues de transferències que apareixen molt seguides, el temps que es trigaria en fer una transferència és un ordre de magnitud més petit en el pitjor dels casos. Mentre que l'enviament es fa en mil·lsegons, els treballs acaben separats per un espai de temps de segons. Aquest anàlisi teòric valida doncs l'estratègia escollida de l'enviament de la informació a través de la xarxa.

### 3.5.2.2 Arquitectura de la solució

Una vegada definida l'estratègia i validada teòricament, és moment des desgranar l'explicació de com assolir els objectius que es plantegen per aquesta part. Com que l'objectiu d'aquesta part és passar dels fitxers temporals amb les dades de rendiment a tenir la informació dins d'una base de dades, es podria plantejar de fer la càrrega directament des dels nodes de càlcul. Tanmateix, es va desestimar aquesta possibilitat per diversos motius. El més important és la sobrecàrrega introduïda als mateixos nodes de càlcul, que repercutiria en un consum de temps als mateixos nodes de càlcul per a realitzar tasques que no són càlcul. Des del *plugin* de SLURM, s'hauria de processar la informació obtinguda i realitzar la traducció del format propi dels fitxers a sentències SQL.

A part d'això, caldria que des dels mateixos nodes de càlcul es pogués accedir a la base de dades, i en cas que hi hagués una ràfega sobtada de treballs acabats es farien moltes connexions simultànies que podrien arribar a provocar algun problema de rendiment tant als nodes de càlcul com al servidor de la base de dades. Seria convenient controlar d'alguna manera el flux de dades per a que la recollida sigui més segura i sense sobrecàrrega sobre els nodes de càlcul.

Basada en la idea de GGCollector, es proposa la solució d'afegir un nou element en la recollida que actui com a intermediari entre els nodes de càlcul i la base de dades. Aquest element serà un nou servidor que s'encarregarà de rebre tota la informació provinent dels nodes de càlcul i realitzar el procés que li calgui. Paral·lelament, haurà d'anar bolcant a la base de dades tota la informació que hagi recollit.

Per tal d'aconseguir aquesta interacció entre els nodes de càlcul i el servidor de recollida, la solució és optar per un model de client-servidor. En aquest cas, el client seria el mateix node (concretament el *plugin* SLURM), mentre que el servidor seria una nova aplicació que quedarà definida posteriorment. El client es posarà en contacte amb el servidor per enviar-li les dades, mentre aquest últim estarà esperant aquestes connexions per a fer la feina.

Ara tot el procés de la informació es realitza fora del supercomputador, pel que la interferència amb el desenvolupament normal de la seva activitat és la mínima. Com a avantatge afegit, aquesta solució permet disposar de més d'un servidor de recollida per augmentar la disponibilitat o realitzar un balanceig de la càrrega.

L'arquitectura proposada també té efectes positius quant a la seguretat. Al aïllar els nodes de càlcul de la base de dades, s'impedeix l'accés en un o altre sentit, ja que la connectivitat entre ells sempre es fa a través d'una màquina intermèdia. D'aquesta forma, el supercomputador i la base de dades poden estar en xarxes diferents, cosa que permet implementar polítiques de seguretat més adients.

### 3.5.2.3 Disseny de pfmcollector

L'aplicació que realitzarà la tasca del servidor de recollida s'anomena pfmcollector. Les seves tasques es poden dividir en diverses parts. Per una banda, s'ha d'encarregar de rebre les dades de cada treball que s'envien des dels nodes de càlcul, per tot seguit tractar-la per a extreure-la d'una manera estructurada. En aquest punt cal definir un sistema de guardat d'aquestes dades que sigui vàlid per al sistema al que s'ha d'integrar. Per últim, haurà d'enviar aquestes dades a la base de dades en forma de sentències SQL. A la figura 3.10 es pot trobar el diagrama de classes de l'aplicació, que s'anirà desglossant a les següents línies.

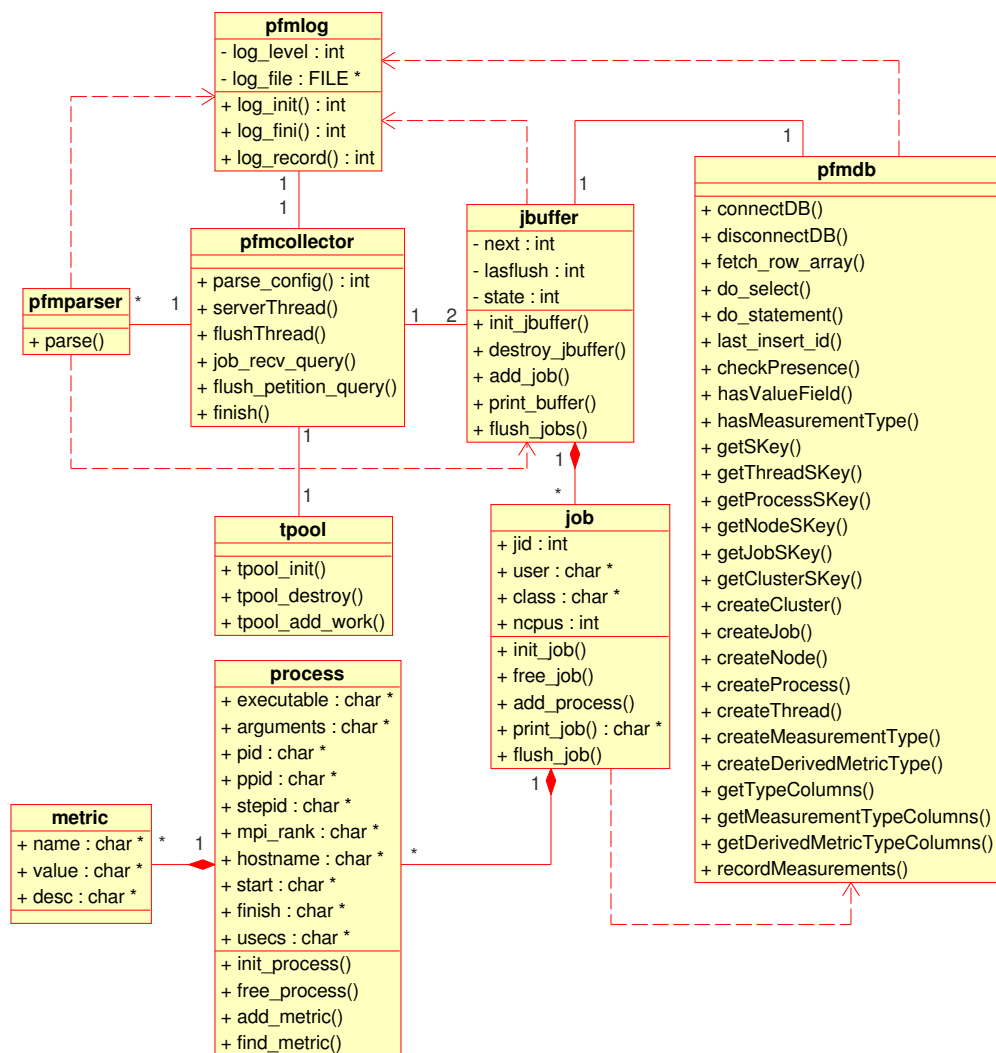


Figura 3.10: Diagrama de classes de pfmcollector

El primer punt és la rebuda de tota la informació. Com que un dels requeriments de la recollida és que els nodes de càlcul no hagin de realitzar esperes innecessàries, el millor és dotar al servidor de paral·lelisme a nivell de *thread*. Aquesta solució, adoptada majoritàriament per qualsevol programa servidor, possibilita tenir un *thread* esperant una nova connexió permanentment. Quan aquesta arriba, se li assigna el treball a un altre *thread*, pel que el primer pot seguir esperant una nova connexió sense esperar que s'acabi de tractar l'actual.

En concret, pfmcollector incorpora el que s'anomena un *thread pool*, o conjunt de *threads* amb aquesta finalitat (tpool). Així, al començar l'execució, es crearan un nombre determinat de *threads*, que quedaran a l'espera de noves connexions. Quan aquesta arriba, un dels *threads* inactius passa a tractar-la, i quan acaba torna a quedar-se disponible per atendre una nova petició. Reutilitzant els *threads* s'evita la sobrecàrrega de la seva creació i destrucció, obtenint un codi més eficient.

Com qualsevol servidor, `pfmcollector` disposa d'un fitxer de configuració (`pfmcollector.conf`), on definir els paràmetres de l'execució del servidor, tals com la mida del *thread pool*, a quin port esperar les connexions, els paràmetres de connexió amb la base de dades o el nom del supercomputador del que es rebrà informació. Amb aquest fitxer es guanya flexibilitat i portabilitat entre supercomputadors, en la que només caldrà adaptar la configuració a les necessitats de cada cas.

Amb la mateixa filosofia s'ha dissenyat un sistema de *log* per al servidor (`pfmlog`). Així, encara que el programa s'executi com a servei, es podrà consultar la seva evolució i detectar possibles problemes de funcionament a través del fitxer generat. Aquest fitxer es pot definir al mateix fitxer de configuració del servidor. En cas que no es defineixi, els *logs* s'imprimiran per la sortida d'error estàndard del programa.

Al mateix temps, es defineixen diversos nivells d'enregistrament, segons la rellevància de l'esdeveniment produït. Amb aquest sistema s'aconsegueix que, des del fitxer de configuració, es seleccioni quin nivell és el desitjat. L'estructura és jeràrquica, pel que cada nivell inclou també la informació dels anterior. Els nivells definits són els següents:

- **Silent (0)** : No es genera cap tipus d'informació.
- **Fatal (1)** : Només els errors fatals (que impedeixen continuar l'execució) són enregistrats.
- **Error (2)** : Es registren tots els errors produïts.
- **Info (3)** : S'escriu informació sobre l'activitat del servidor. És el nivell recomanat.
- **Debug (4)** : El mode més complet, escriurà molta més informació per a depurar el programa. Aquest mode només s'ha d'utilitzar en aquest cas, ja que el fitxer que s'escriu creixerà ràpidament amb cada acció.

El mètode escollit per a la comunicació entre client i servidor és per mitjà de *sockets* TCP. D'aquesta manera es redueix la sobrecàrrega que s'introduiria amb altres protocols com FTP, SSH o RPC, però presenta alguns punts febles. El principal és que no es produeix cap autenticació per a l'enviament, ni es xifren les dades enviades. Això podria ser un problema de seguretat, però queda atenuat pel fet que tota la comunicació esdevé en un entorn tancat i controlat. Per a implemencions futures, es podria pensar en utilitzar el protocol de crides remotes RPC per a fer-lo més portable i funcional.

El protocol de comunicació seguit és el més senzill possible. El client (*plugin* SLURM) envia una petició d'enviament al servidor indicant quin treball vol enviar. Tot seguit, si el servidor està

disponible, li respondrà positivament i es prepararà per a rebre les dades que el client li enviarà a continuació. Si el servidor no està disponible, el client intentarà el mateix amb un altre servidor dels que tingui definits.

Un cop es reben les dades en el format original, és moment de pensar què s'ha de fer amb elles. Per agilitzar el màxim possible el funcionament d'aquest servidor, s'evitarà utilitzar el disc per emmagatzemar temporalment el que arriba per la xarxa. El que farà és guardar-ho tot a memòria, que té una velocitat d'accés molt més alta. Utilitzant aquesta solució cal tenir en compte que el consum de memòria del servidor estigui dins dels paràmetres raonables. També cal notar que en cas que el programa finalitzi inesperadament, tot el que no hagués bolcat a la base de dades es perdria irremediablement.

S'ha dissenyat una estructura de dades per a poder emmagatzemar de manera organitzada tota la informació que arriba. Es pot observar a la figura 3.10 aquesta estructura, anomenada *jbuffer*. Com el nom indica, és un *buffer* circular que contindrà informació dels treballs, guardats de forma consecutiva. Per a acotar d'alguna manera la memòria consumida, aquest vector serà de mida fixa. Com a conseqüència d'això, es pot arribar a perdre a informació en el cas que s'omplís el *buffer* abans de fer un bolcat a la base de dades, sobreescrivint entrades de treballs que encara no havien estat salvats.

La forma d'evitar aquest problema és realitzar el bolcat amb la freqüència necessària per tal que no s'arribi a omplir mai el *buffer*. La decisió de disseny és doncs la de sacrificar la obtenció completa d'informació a canvi de garantir un control sobre la quantitat d'informació temporal emmagatzemada, ja que la informació d'un treball concret és prescindible. Aquesta idea també s'aplica en altres sistemes de recol·lecció a gran escala com *GGCollector*.

L'estructura de *jbuffer* és jeràrquica, i té com a element principal el treball. Cada treball tindrà una informació associada a la totalitat del treball (com l'identificador del treball o la classe), i el compondran una sèrie de processos o *threads*. A la seva vegada, cada procés tindrà associada informació identificativa del mateix a més d'una sèrie de mètriques de rendiment. Cal notar que en aquest punt no es distingeix entre procés i *thread*, ja que la informació guardada serà la mateixa a excepció de l'identificador de *thread*, que es computarà com una mètrica més.

Quan el servidor comença a rebre les dades d'un treball, ha de tractar-les tal com arriben per agilitzar el procés. El tractament consisteix en llegir cada línia del que arriba per la xarxa i col·locar-ho a l'estructura de dades anterior. Fent això, s'evita haver d'emmagatzemar temporalment les dades rebudes i es guanya velocitat de resposta. Cal recordar que aquesta tasca és realitzada per un *thread* dedicat, mentre que el servidor segueix disponible per atendre noves peticions.

El servidor no només pot rebre peticions per a rebre informació, sinó que també està preparat per rebre ordres de realitzar un bolcat de les dades que tingui guardades. Aquest procés, que també rep el nom de *flush*, és una altra de les tasques principals del programa. Es pot activar mitjançant una ordre exterior o bé periòdicament. Des del fitxer de configuració es pot establir l'interval de temps entre cada bolcat, que cal ajustar adequadament al volum de dades que s'espera recollir per tal d'obtenir un flux raonable de peticions cap a la base de dades i al mateix temps tenir cura de que el *buffer* no s'arribi a omplir completament.

A la figura 3.11 es pot comprovar el procés dissenyat per a realitzar el bolcat. Per facilitar la comprensió, s'ha simplificat el diagrama per només mostrar els punts realment importants. El bolcat s'executa quan es rep una petició per fer-ho, i també perquè s'ha consumit l'interval de temps definit entre bolcats. En qualsevol cas, es farà una connexió a la base de dades, i després per a cada treball dels que són al *buffer* generarà les sentències SQL necessàries i les executarà contra la base de dades. En un mateix treball, hi haurà la informació de diversos processos i *threads*, amb les seves mètriques associades que també caldrà inserir.

En el moment de realitzar el bolcat del *buffer*, cal que aquest es mantingui sense canvis durant tot el procés. Això vol dir que mentre el *flush* és en marxa, no es pot afegir cap treball nou al *buffer*. Això planteja un problema de disponibilitat, perquè aquesta tasca pot durar de l'ordre de segons, i en aquest temps és molt probable que es rebin noves peticions. La solució a aquest problema ha estat dotar al servidor de doble *buffer*.

En tot moment sempre hi haurà un dels *buffers* que estarà rebent tota la informació, mentre que l'altre restarà inactiu. En el moment que s'ha de realitzar un bolcat, es canvia de *buffer*. El que estava rebent els treballs fins ara passa a estar inactiu i l'altre ocupa el seu lloc. D'aquesta manera, es pot realitzar tot el bolcat sense que el *buffer* pateixi modificacions, ja que les noves peticions són absorbides per l'altre. Un cop acabat el *flush*, el *buffer* estarà buit i inactiu, esperant a que arribi una nova petició de bolcat, que tornarà aquest *buffer* a la primera línia i repetir tot el procediment.

Per últim, *pfmdb* implementa la interacció entre el servidor i la base de dades, oferint una interfície simple al primer per a crear, consultar i inserir noves dades com el treball, els processos o les seves mètriques. En aquest punt s'ha adaptat la interfície original de PerfMiner, que feia aquesta feina mitjançant un *script* Perl, per a poder treballar amb la resta de components del servidor de recollida.

### 3.5.3 Implementació i instal·lació

Per a la implementació de *pfmcollector* s'ha utilitzat el llenguatge C. La raó principal és que l'eficiència d'aquest component del sistema és molt important, així com el seu consum de memòria

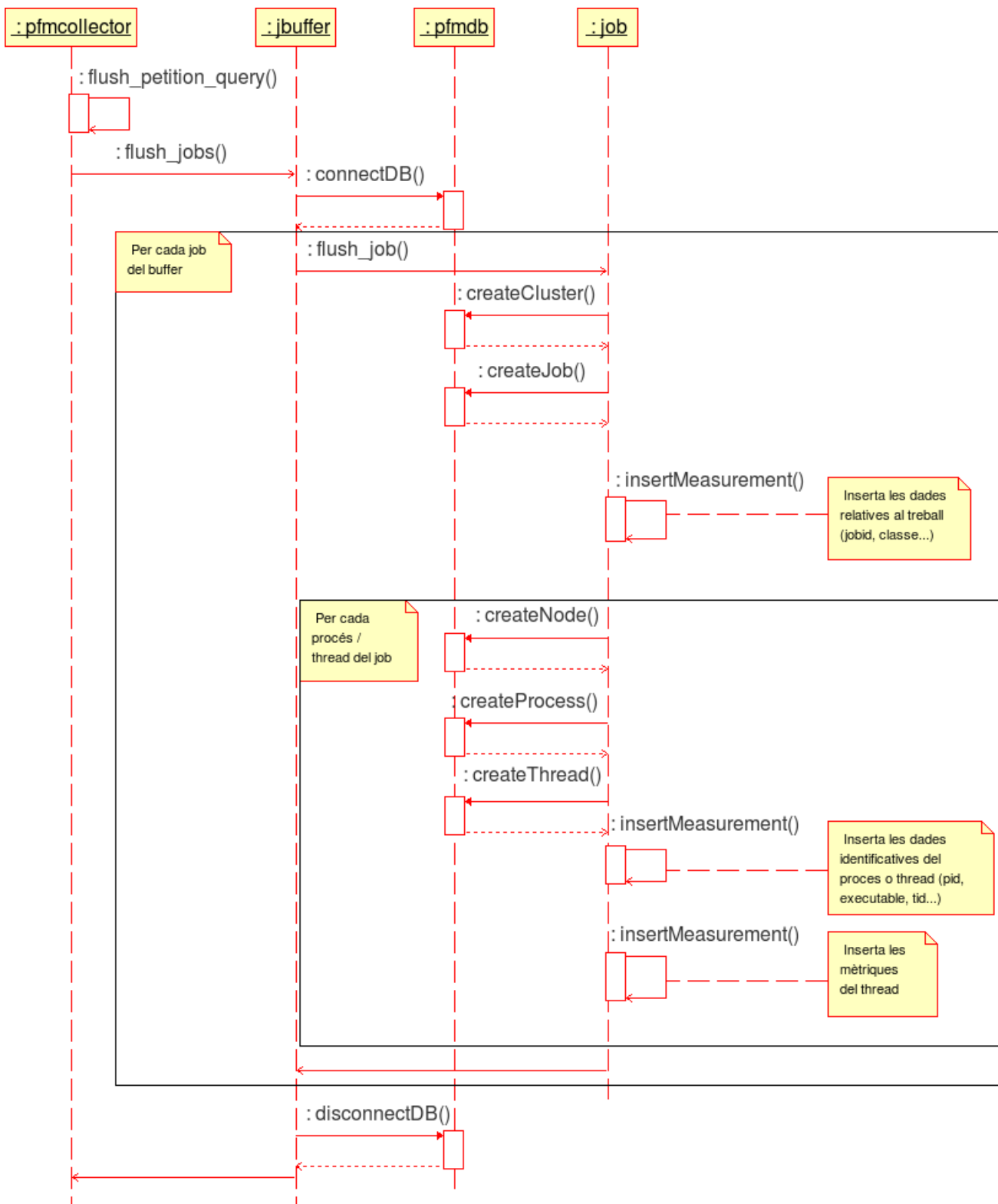


Figura 3.11: Diagrama de seqüència del procés de *flush*

i recursos. El llenguatge C ofereix un gran control al programador de l'ús dels recursos, però per contra s'obtenen programes menys modulars i més difícils de mantenir. S'ha sacrificat doncs la portabilitat i la mantenibilitat per l'eficiència, tot i que per a versions futures, després d'avaluar el rendiment de l'actual, es podria pensar en una implementació més modular, orientada a objectes, en C++.

Com al mateix *plugin* SLURM, s'ha fet ús de les llibreries *etools* per aprofitar característiques com llistes dinàmiques o taules *hash*, que ni el llenguatge C ni les seves llibreries estàndard no ofereixen. També s'ha implementat la lectura de fitxers de configuració amb aquestes llibreries. Igualment, s'ha utilitzat la API C del client de MySQL[MyS06] per a realitzar les operacions contra la base de dades. Per a versions futures, caldria pensar en reimplementar aquesta part utilitzant una API més genèrica, que permeti major portabilitat entre sistemes gestors de bases de dades diferents.

El resultat de compilar el codi és un executable que pot ser carregat des de qualsevol lloc (ja que és autocontingut), amb la única condició de que el fitxer de configuració estigui al mateix directori. S'ofereix també el fitxer de configuració amb els paràmetres estàndard i amb comentaris per ajudar a l'hora de posar els valors més adients.

Per assegurar el funcionament correcte quant a l'ús de la memòria, s'ha utilitzat la utilitat Valgrind[SN], que permet detectar, entre d'altres coses els anomenats *memory leaks*. De produir-se aquest problema, el programa aniria consumint cada cop més memòria a mida que passa el temps, ja que no allibera tota la que no li cal. Com que aquest és un programa que estarà dies funcionant, interessa que aquest aspecte estigui totalment controlat.

## 3.6 Emmagatzemament a la base de dades

A la secció anterior s'ha explicat el sistema de recollida de la informació, però no s'ha entrat en detall de què fer-ne un cop recollida. En aquest apartat es descriu l'emmagatzemament de la informació que s'ha obtingut. El primer serà veure quins requisits són els més significatius d'aquesta part del sistema, per després analitzar el disseny que s'ha utilitzat i la seva integració.

### 3.6.1 Requisits principals

Per a poder realitzar un anàlisi del comportament de les aplicacions i del supercomputador en general, queda clar que s'ha d'oferir un suport permanent per a les dades recollides, més enllà dels

espais temporals que fins ara s'han utilitzat. El millor per fer això és una base de dades on es pugui emmagatzemar de manera estructurada la informació i que després permeti desenvolupar consultes i extreure conclusions globals.

Aquesta base de dades haurà d'acollir dades de rendiment a diferents resolucions (des del clúster fins al *thread*). Les dades no seran més que mètriques amb un cert valor, i podran ser diferents segons el nivell d'extracció d'informació que s'hagi escollit en cada cas. Cal per tant una base de dades flexible, que permeti fàcilment acomodar tot tipus de mètriques relacionant-les amb el seu referent.

Al mateix temps, no només les mètriques poden variar sinó que també ho pot fer el seu nombre. Cal un model que sigui fàcilment extensible a noves mètriques que es puguin extreure, on inserir la nova informació sense haver de modificar l'esquema inicial. De la mateixa manera, molts cops interessarà realitzar un cert anàlisi sobre mètriques derivades de les originals. Aquesta base de dades hauria de ser capaç de donar suport a les mètriques derivades que es puguin obtenir d'altres que ja estiguin presents, i de manera fàcil i automàtica.

Un punt important també a l'hora de dissenyar la base de dades és tenir en compte la quantitat d'informació que s'hi emmagatzemarà. Com que la informació provindrà de grans supercomputadors, el sistema gestor de bases de dades haurà de treballar amb taules de milions d'entrades a cada consulta. Tot i que el temps de resposta de les consultes no és crític, cal una base de dades escalable a aquesta quantitat d'informació.

### 3.6.2 Disseny de la base de dades

Com en seccions anteriors, el primer pas ha estat analitzar la solució proposada per PerfMiner. A diferència de parts com la integració amb el sistema o la recollida, aquesta sí que s'adapta quasi perfectament al projecte actual. De fet, el seu disseny es basa en els mateixos principis enunciats a l'apartat anterior. Com que el projecte partia de la premissa d'integrar en la mida del possible eines i programari ja existent, la decisió ha estat integrar l'esquema original al nou sistema.

Aprofitar el disseny original proporciona diversos avantatges. Els més evidents són els propis de la reutilització de programari, com són la d'aconseguir un sistema més estable i provat des del primer moment. A aquest fet s'afegeix la possibilitat de poder aprofitar també l'eina de visualització i anàlisi que treballa sobre aquesta base de dades. Aquesta part però serà tractada a la secció 3.7.

El disseny de l'esquema de la base de dades es mostra a la figura 3.12. Com que la base de dades ha de ser flexible i extensible, el seu disseny és totalment modular, pel que l'esquema que es pot

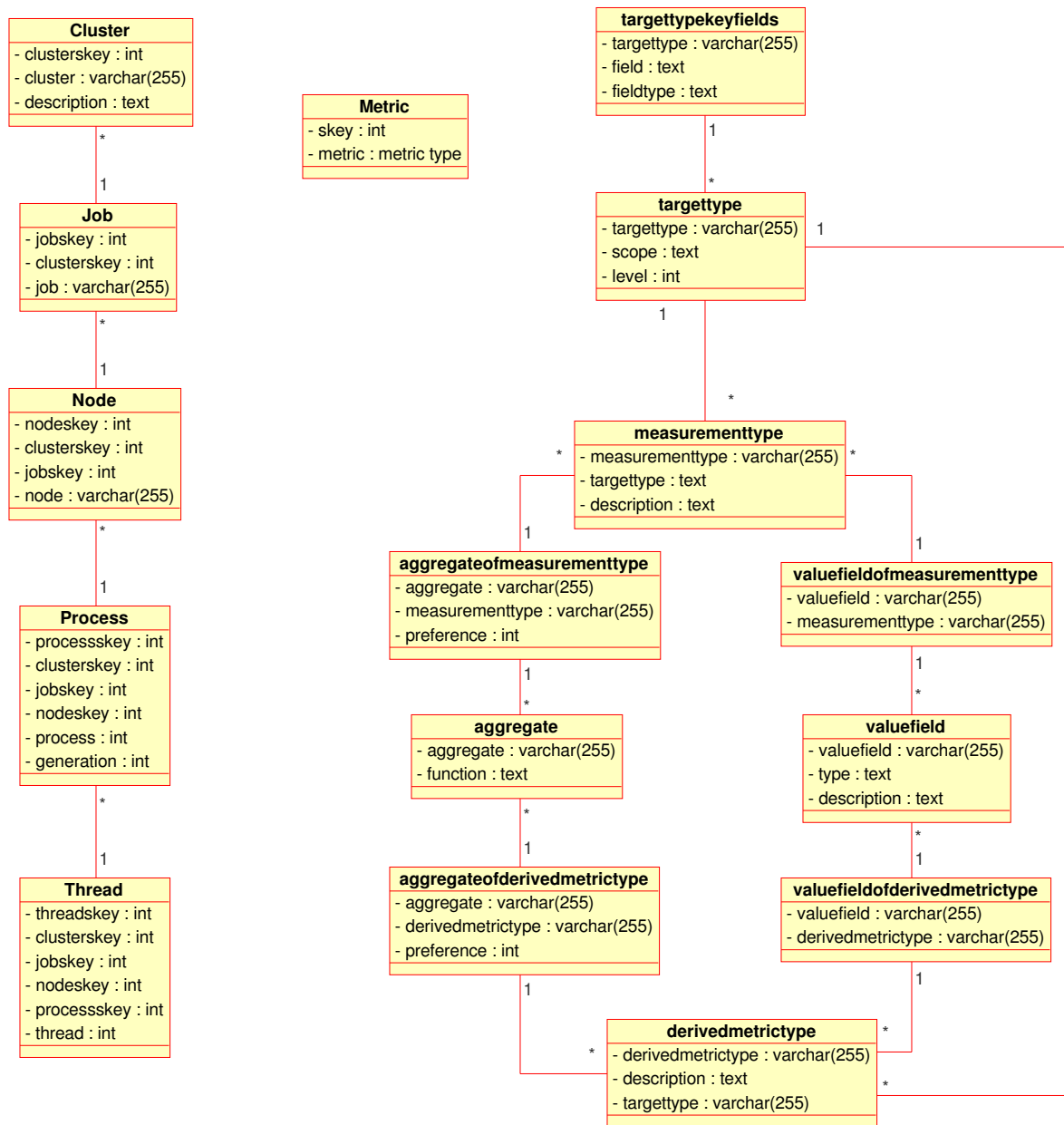


Figura 3.12: Esquema de la base de dades

veure aquí no és l'esquema complet, sinó que és només la base a partir de la qual es construeix la resta de la base de dades.

Es parteix d'una jerarquia d'elements que poden relacionar-se amb les mètriques. Aquests objectius (o *targets*), que apareixen a la part esquerra de la figura 3.12 són el clúster, el treball (*job*), el node del clúster, el procés i el *thread*. Com es pot veure, cada categoria inclou l'anterior: així un *thread* pertany a un procés, que s'ha executat a un cert node assignat a un determinat treball del clúster. Aquesta estructura facilita l'agregació de les dades obtingudes a diferents resolucions.

Cada tipus de mètrica que s'obtingui formarà per si sola una nova taula. És per això que aquestes no figuren a l'esquema, ja que el nombre de mètriques és totalment indeterminat. Això queda reflectit a l'esquema mitjançant la taula *Metric*. una taula genèrica que simbolitza totes aquestes taules indeterminades. Aquesta solució és totalment flexible, ja que es poden afegir noves mètriques simplement creant una nova taula per a elles, i no implica la modificació de cap taula.

Tot i això, com que cada mètrica és diferent, cal guardar certa informació relacionada amb aquestes mètriques per a després poder interpretar-la correctament. Aquesta és la funció que duu a terme la part dreta de la figura 3.12. Concretament, es manté informació de quin és l'àmbit de la mètrica (relacionada amb procés, relacionada amb treball, etc), així com també les funcions d'agregació que se li poden aplicar. Les funcions d'agregació disponibles van des de la simple suma fins al càlcul de mitjanes o desviacions estàndard.

Aquesta base de dades també ofereix suport a les mètriques derivades, que es definiran com una vista. Així, a l'hora de fer les consultes serà el mateix tractar amb mètriques derivades o amb mètriques reals. D'aquestes també es guardarà les mateixes dades: el tipus d'informació que emmagatzema i les funcions que se li poden aplicar.

Per últim, cal destacar la utilització de claus alternatives (anomenades sintètiques) per als diferents nivells d'agregació. Amb aquesta tècnica s'evita que als nivells inferiors, com el de *thread*, s'arrossequin totes les claus dels nivells superiors i per tant sigui més fàcil crear índexs sobre aquestes taules. La creació d'índexs és un procés important, ja que s'haurà de treballar amb operacions de *join* sobre taules que poden arribar a tenir milions d'entrades. Si no es disposen d'índexs sobre les claus primàries de les taules, aquestes operacions poden arribar a ser molt costoses ( $O(n^2)$ ). En canvi, amb la utilització dels índexs es pot arribar a reduir el temps fins a  $O(n \log n)$  segons l'algorisme que el sistema gestor utilitzi.

### 3.6.3 Integració i adaptació

La base de dades original de PerfMiner està pensada per treballar sobre PostgreSQL<sup>3</sup>. En canvi, per disponibilitat al centre, s'ha utilitzat MySQL en la seva versió 5.0 per allotjar-la. Tot i que en principi la base de dades hauria de ser compatible amb qualsevol sistema que compleixi amb l'estàndard SQL95, a la pràctica apareixen certs problemes que cal tractar.

El primer canvi fet és sobre els tipus de molts camps. Molts d'ells tenien tipus text, que és de mida indefinida. Això no és problema en la majoria dels casos, excepte en aquells que actuen com a clau. En aquests casos, MySQL obliga a acotar la mida del tipus d'aquells camps que actuen com a clau, pel que es van haver de canviar els tipus text per un varchar acotat (255 caràcters en la majoria dels casos) allà on calia.

Per altra banda, MySQL no permet realitzar operacions de *join* amb taules temporals, pel que moltes de les consultes s'han hagut de reescriure evitant utilitzar aquestes tècniques. El cas més crític és el d'una vista de la base de dades anomenada *metric*, que aglutina en una mateixa taula informació de totes les mètriques que es guarden a la base de dades. Aquesta és utilitzada freqüentment des de l'aplicació de visualització per la facilitat d'obtenir totes les metadades de cada mètrica.

També per afegir de més funcionalitat a la base de dades i les eines d'anàlisi que hi treballaran, s'ha creat una nova funció d'agregació anomenada *load balance*, o balanceig de la càrrega. Aquesta funció realitza una mitjana dels valors de cada conjunt, i després la divideix pel màxim trobat. Així, aquesta funció es pot aplicar sobre diverses mètriques i permet representar d'una forma fàcil l'equilibri en la distribució dels valors de la mètrica entre els diversos elements.

MySQL permet definir funcions d'usuari com la descrita mitjançant la creació d'una llibreria dinàmica que la implementa. Aquesta llibreria, escrita en C, utilitza una API que ofereix el mateix servidor i que la carregarà en el moment de ser utilitzada, passant-li els valors sobre els que ha de fer el càlcul.

Per altra banda, s'ha aprofitat la possibilitat que el mateix disseny ofereix per a crear noves mètriques derivades a partir de les ja existents. Un exemple seria una nova mètrica per caracteritzar millor la taxa de fallades de la memòria cau L2, que consisteix en calcular la mitjana d'accessos fallits a aquesta memòria cau per a cada mil instruccions executades. Com es pot veure, per a la seva construcció s'utilitzen mètriques ja existents com el nombre de fallades de la cau L2 o el nombre d'instruccions.

---

<sup>3</sup>Veure <http://www.postgresql.org/>

Donades les característiques singulars de la base de dades, s'ha hagut d'adaptar la configuració del servidor de bases de dades, ja que quan les mides de les taules augmenten, es pot arribar a col·lapsar pels petits *buffers* temporals definits. A més, per motius de seguretat, el servidor que proporciona la base de dades no té connexió directa des de l'exterior del supercomputador, i cal realitzar túnels per accedir-hi (ja sigui via SSH o utilitzant les regles dels diversos tallafocs).

## 3.7 Anàlisi i visualització

A continuació es tracta l'última part del sistema, que no és altra que la de la visualització de les dades obtingudes que resten allotjades a la base de dades descrita a la secció anterior. Com en apartats anteriors, es partirà d'una descripció dels requisits principals que s'exigeixen en aquesta part, per seguir amb els casos d'ús típics del sistema. Un cop vistos, es veurà el disseny de la interfície web i la implementació que s'ha dut a terme.

### 3.7.1 Requisits principals

El sistema que s'ha desenvolupat és capaç de generar i emmagatzemar grans quantitats de dades, que també són a la vegada molt heterogènies. La base de dades segueix un disseny que s'adapta a aquesta heterogeneïtat, ja que permet fàcilment ampliar el tipus d'informació que es guarda. Aquesta mateixa flexibilitat i extensibilitat és la que s'hauria d'aconseguir en aquesta part, per a poder visualitzar les dades amb la mateixa facilitat que s'han emmagatzemat.

La flexibilitat i extensibilitat a la base de dades tenen un preu, i és la complexitat en la que resulta la seva estructura i per tant la dificultat de realitzar consultes per extreure'n la informació. La part de visualització ha de facilitar aquesta feina, oferint una interfície el més simple possible per a l'anàlisi de les dades i que amagui la complexitat intrínseca que representa cada consulta. Per això s'han d'utilitzar gràfics i taules que mostrin les dades, facilitant-ne la comprensió i l'anàlisi.

Igualment, la base de dades relaciona cada dada de rendiment amb una jerarquia d'elements que actuen com a objectius. L'eina de visualització ha d'aprofitar aquest fet per a poder obtenir informació agregada als diversos nivells de la jerarquia, utilitzant a l'hora les metadades que també estan emmagatzemades per a cada mètrica.

És desitjable també, seguint en la línia de facilitar la feina a la persona que ha d'analitzar les dades, que aquesta part sigui fàcilment accessible. L'analista no hauria d'instal·lar cap aplicació ni

fer un llarg procés per a començar a treballar, sinó que ha de tenir una manera ràpida d'accedir a l'aplicació de visualització i des de qualsevol lloc.

### 3.7.2 Disseny de la interfície web

Com en altres parts del projecte, el primer pas és veure què ofereix PerfMiner i si s'adapta a les necessitats concretes del sistema en desenvolupament. PerfMiner utilitza per a la visualització de la informació recollida una interfície web mitjançant la qual es poden generar taules i gràfiques amb les dades obtingudes, a diferents resolucions, partint de l'oportunitat que ofereix el disseny de la base de dades.

Aquesta interfície destaca per la seva simplicitat a l'hora de treballar, amagant en bona part la complexitat de la base de dades. A més, el fet d'estar disponible a través del web també és un punt a favor, ja que la persona que analitza les dades no ha d'instal·lar cap tipus de programari, i pot accedir i començar a treballar a l'instant des de qualsevol equip amb connexió.

Aquests aspectes satisfan perfectament els requisits exposats a l'apartat anterior, pel que la decisió és prendre com a referència la solució original per a aquesta part del projecte. Cal recordar que al respectar també l'essència del disseny original de la base de dades, la integració de la interfície original resulta molt més simple.

Com es pot veure a la figura 3.13, la part de la visualització està formada per diversos components. El paper del model aniria a càrrec de PerfminerDB, que conté totes les operacions relacionades amb la interacció amb la base de dades. Posteriorment, la vista estaria representada per la classe Graphs, que és l'encarregada de realitzar els gràfics amb les dades obtingudes. Per últim, Perfdata actuaria de controlador rebent les requestes, construint el resultat mitjançant PerfminerDB i resrepresentant-lo amb Graphs.

Un dels punts que destaquen de la proposta de PerfMiner és que bona part de la complexitat del càlcul queda en la pròpia base de dades, ja que es realitza l'agregació pertinent de les dades a nivell de la pròpia consulta. Gràcies a la informació addicional que es manté a la base de dades sobre les diferents mètriques, s'afavoreix un disseny molt genèric dels components de visualització, i en concret sobre la classe que actua de model, PerfminerDB.

Amb les opcions seleccionades, es construeix la consulta i els resultats queden guardats dins l'objecte PerfminerDB, concretament utilitzant PfmResult, per tal que després es puguin representar de la manera que calgui, ja sigui amb gràfics o taules. Per a completar els resultats, també es manté informació de context de les dades, com els noms de cada element (metadades), mitjançant Pfm-

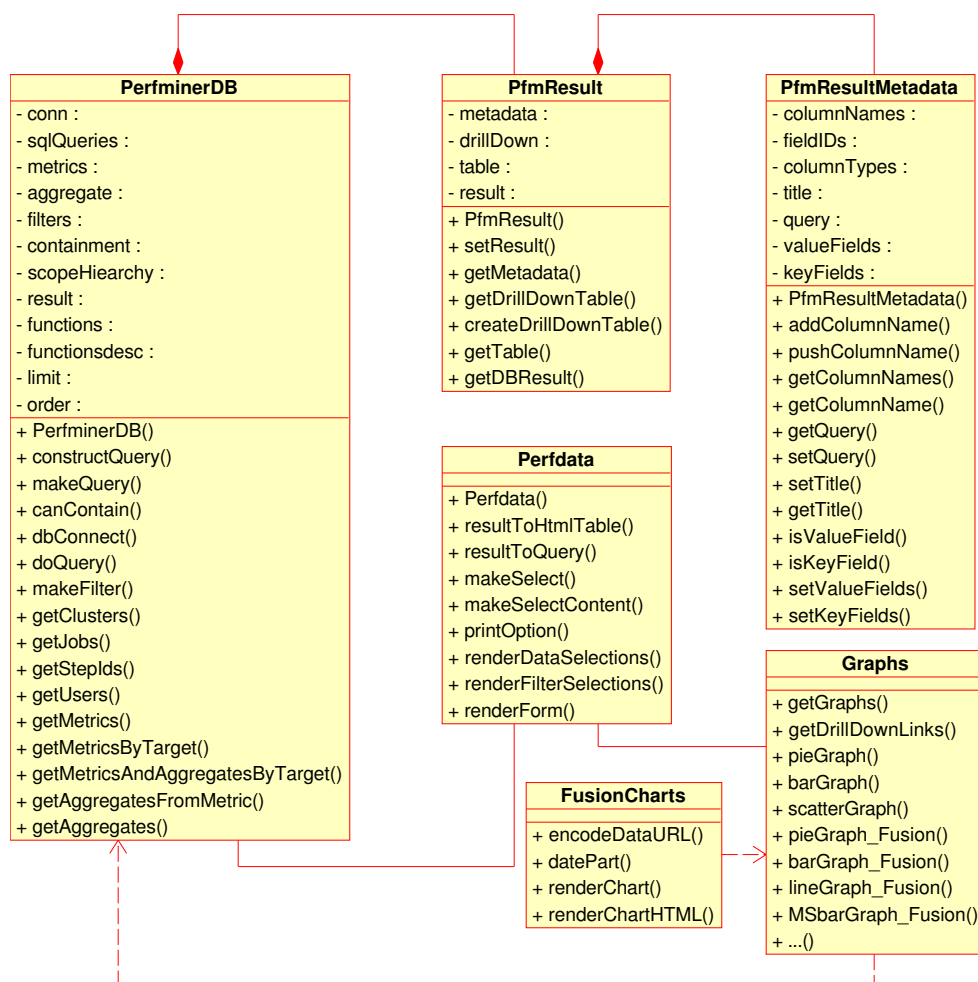


Figura 3.13: Diagrama de classes de la interfície web de PerfMiner

ResultMetadata. També s'emmagatzema la consulta en si per si més endavant es vol visualitzar com a mode de depuració del programa.

Pel que fa la presentació de la informació, l'aplicació consta d'una plana web amb tres parts diferenciades. En primer lloc, apareixen una sèrie d'elements que permeten acotar la consulta que es fa. Exemples d'aquesta part són la selecció de l'usuari, el treball o el clúster. Posteriorment, es troba la part d'opcions de visualització, on es permet especificar quina informació es mostrarà (amb un llistat de mètriques) i com es mostrarà (ja sigui amb un gràfic, una taula, etc). Per últim, es troba la visualització de la informació d'acord amb els paràmetres especificats a les altres dues parts. El disseny permet que es mostri el mateix resultat amb mètodes diferents, com per exemple un gràfic i una taula alhora.

### 3.7.3 Adaptació i millora

Durant el desenvolupament d'aquesta part, s'ha col·laborat estretament amb l'autor original, Daniel Ahlin, per aconseguir una versió millorada i que satisfés les necessitats de totes les parts. En el marc d'aquesta col·laboració el desenvolupament actiu que s'ha dut a terme en aquesta part del projecte s'integra dins la versió oficial de l'eina PerfMiner.

Per a la implementació d'aquesta interfície web, la solució original utilitza PHP com a llenguatge de programació i, lògicament, HTML. Per a la creació de gràfics, s'utilitza l'eina JGraph[Per], que permet dibuixar dinàmicament gràfics de barres, circulars, etc. Com que la implementació original utilitza PostgreSQL com a base de dades, la interacció des de la web es produeix amb la API de PHP per a PostgreSQL.

El primer punt en el que apareixen conflictes entre la versió original i la de l'actual projecte és precisament en la base de dades. Com que en l'actual s'utilitza MySQL, la API a utilitzar és diferent i no compatible amb la de la implementació original. En una primera versió, es va migrar tota la interacció amb la base de dades a la API de MySQL.

Tot i funcionar bé, no és una bona alternativa de cara al manteniment d'una única versió per als dos sistemes. Per això es va optar per utilitzar una API que permetés abstraure el sistema gestor de bases de dades de forma que no calgués canviar el codi d'aquesta part en funció de la base de dades utilitzada. Això s'ha aconseguit gràcies a la classe de PHP anomenada PDO (PHP Data Object). Aquesta classe conté una sèrie d'operacions genèriques per aconseguir dades des d'una base de dades, i internament gestiona les diferents implementacions per a diferents sistemes de bases de dades.

L'assortit de gràfics que ofereix la versió original és força limitat, pel que s'incorpora una nova tecnologia de gràfics, FusionCharts Free[Fus]. Aquesta tecnologia produeix gràfics en format flash, i la informació a representar se li passa en forma de fitxer XML, pel que resulta molt senzill crear un gràfic amb les dades obtingudes de la base de dades. A més de ser visualment molt atractius, existeixen molts tipus de gràfics diferents que es poden incorporar al sistema.

El disseny de la representació de la informació permet afegir noves formes de gràfics sense grans canvis en el codi, sinó simplement definint una nova funció per inicialitzar el gràfic dins la classe Graphs. El sistema fa la resta i incorpora el nou gràfic a la llista de possibles representacions.

En la versió original de l'aplicació, l'usuari pot elegir quina mètrica o mètriques vol obtenir a una certa resolució, i el sistema s'encarrega de l'agregació al nivell corresponent. Com que cada mètrica té unes funcions d'agregació definides, i una preferència, el sistema sempre escull aquella

funció amb més preferència. Tot i això, no sempre es vol utilitzar la mateixa funció per a una mètrica concreta, o fins i tot seria desitjable agregar en una mateixa vegada una mètrica utilitzant diferents funcions.

La proposta del projecte consisteix en una funcionalitat opcional, que en el moment d'activar-se, mostra per a cada mètrica les funcions disponibles d'agregació. Sel·leccionant les desitjades, el sistema mostrarà els resultats utilitzant les funcions seleccionades en cada mètrica en comptes de la que apareix com a preferida. Si la opció no està activa, el sistema es comportarà com l'original.

Seguint amb la configuració dels resultats, quan la base de dades creix amb l'ús, algunes consultes poden produir milers i fins i tot milions de resultats. El cost tant d'aquesta consulta com de la seva representació pot ser molt gran i pot també dificultar l'anàlisi de la informació. Per això s'introdueix la possibilitat de limitar els resultats que es volen obtenir a un cert nombre. Com en el cas anterior, si no s'especifica res, el comportament és idèntic al sistema original.

Per augmentar la flexibilitat, també es proporciona una nova opció per ordenar els resultats. Com que les dades a representar és altament variable, el que s'ofereix és la possibilitat d'ordenar tots els resultats d'una manera ascendent o descendent. Aquesta funcionalitat pot resultar molt útil combinada amb l'anterior limitació del nombre de resultats, ja que depenent de com s'hagi efectuat la limitació, es poden quedar fora resultats que podrien haver resultat interessants.

En la versió original de l'aplicació, pot resultar una mica difícil l'anàlisi d'una execució paral·lela independentment del seu context del treball. Per això es proposa la creació d'una nova possibilitat de sel·lecció amb l'aparició de StepId. Com que cada StepId representa una execució paral·lela dins un treball, pot resultar interessant per analitzar el comportament de l'aplicació paral·lela corresponent. Amb aquesta sel·lecció, es descarta tant la part seqüencial i d'altres execucions paral·leles del mateix treball. Per combinar-ho amb aquesta funcionalitat, també s'afegeix una nova possibilitat de resolució a les que ja existeixen, que és el MPI\_Rank.

També s'ha millorat la possibilitat de realitzar *drill-down* de les dades obtingudes, o el que es el mateix, baixar en els nivells de resolució per a obtenir uns resultats cada cop més acotats. Això es permet gràcies als enllaços que queden generats tant a les taules com als gràfics. Si es pressiona sobre qualsevol entrada, es generarà un nou resultat amb les mateixes característiques que l'anterior però a un major nivell de detall. Per exemple, si en un determinat moment els resultats que apareixen per pantalla són els MFLOPS per a cada treball, i es pressiona sobre un treball, el nou resultat seran els MFLOPS d'aquell treball agrupats per node (el següent element en la jerarquia de resolucions).

L'aspecte de l'aplicació web és el que es pot observar a la figura 3.14. Es distingeixen les parts mencionades en aquests últims apartats: la part superior, on es permet realitzar seleccions sobre la informació que es vol obtenir, la part on es pot triar què es vol representar i com, i per últim la part de la representació en si.

## BSC Performance Miner

**Clusters**

ALL  
nord

**Users**

ALL  
xabellan  
jnarango  
dvicente  
smore  
dguardia  
afont

**Jobs**

ALL  
nord-60727  
nord-60726  
nord-60725  
nord-60723  
nord-60722  
nord-60721  
nord-60719  
nord-60718  
... (562 more)

**Steps**

ALL  
nord-60727-1  
nord-60726-1  
nord-60725-1  
nord-60723-1  
nord-60722-1  
nord-60721-1  
nord-60719-1  
nord-60718-1  
nord-60717-1

**Resolution**

cluster  
job  
node  
process  
thread  
Program name  
MPI Rank  
Job User

**Metrics**  Show aggregate functions

- L2 Instruction Cache Accesses (total)
- L2 Instruction Cache Misses (average)
- L2 Instruction Cache Misses (max)
- L2 Instruction Cache Misses (min)
- L2 Instruction Cache Misses (standard deviation)
- L2 Instruction Cache Misses (total)
- L2 Miss Ratio (average)**
- L2 Miss Ratio (max)**
- L2 Miss Ratio (min)
- L2 Miss Ratio (standard deviation)

Rock on!

**Outputs**

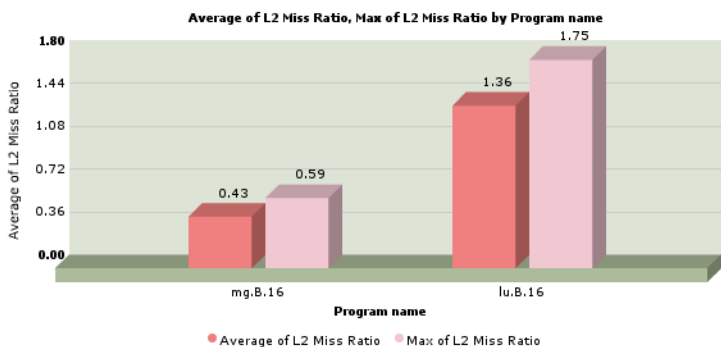
- Pie Chart(s)
- Bar Chart(s)
- Scatter Plot(s)
- Pie Chart 3D
- Bar Chart 3D
- Line Graph
- Multiple Bar Chart**
- Table
- Query

**Limit of Rows**

1000

**Order of Rows**

Ascendant  
**Descendant**



Program name	Average of L2 Miss Ratio	Max of L2 Miss Ratio
<a href="#">/gpfs/home/Operacions/xabellan/NPB2.4/NPB2.4-MPI/bin/mg.B.16</a>	0.43263204394401	0.58930104979114
<a href="#">/gpfs/home/Operacions/xabellan/NPB2.4/NPB2.4-MPI/bin/lu.B.16</a>	1.3632687235953	1.7479059860719



Figura 3.14: Vista principal de l'aplicació web per a la visualització



## Experiments i Resultats

Havent detallat ja tots els components del sistema, vist el seu disseny i com s'ha dut a terme la implementació, és moment de validar empíricament el sistema obtingut. Com que el sistema està compost per diverses parts, s'han realitzat diferents proves destinades a comprovar tots els aspectes rellevants del seu funcionament. A més, per a aquest propòsit s'han utilitzat diferents entorns de proves, tant a l'hora del desenvolupament com en fases posteriors.

A continuació s'explicaran els diferents entorns de proves utilitzats durant la realització d'aquest projecte, per seguir posteriorment amb els experiments realitzats per validar el correcte funcionament del sistema desenvolupat.

### 4.1 Entorns de treball

Per a cada fase del projecte i cada part s'ha utilitzat un entorn de proves i de treball diferent i adequat a les necessitats de cada moment. Aquests són els entorns utilitzats, amb la seva descripció i els usos que se'ls hi han donat.

#### 4.1.1 Ordinador portàtil estàndard

Aquesta és la primera plataforma on s'ha començat a desenvolupar el projecte. És un portàtil IBM Thinkpad T60, amb un processador Intel Core duo de doble nucli. Aquest portàtil ha estat la base d'operacions des de la qual s'ha treballat durant el projecte, ja sigui per fer feina localment o per connectar-se a les altres màquines.

El portàtil ha estat la plataforma on s'han fet els primers desenvolupaments de parts com el *plugin* SLURM, *pfmcollector* o la part de visualització. Al mateix temps, ha servit per a realitzar les primeres proves i validar el funcionament per als casos més senzills. Amb aquest propòsit es va instal·lar SLURM i es va configurar un sistema de cues amb només un node (el mateix portàtil). Això va permetre instal·lar també el *plugin* i depurar els errors que van sorgir a les primeres versions. Fent-ho localment no s'interfereix en sistemes que estan en producció o semi-producció i s'evita causar problemes als usuaris.

De fet, el mateix portàtil pot allotjar el sistema complet, ja que gràcies a la instal·lació del servidor de bases de dades MySQL i el servidor web apache, la mateixa màquina pot representar tots els rols de cada part del sistema. De tota manera, l'entorn de treball del portàtil és molt diferent del que es pot trobar en entorns més reals, i per tant és un banc de proves limitat. Serveixi com a exemple de la limitació l'existència d'un sol node en el sistema, o la impossibilitat de realitzar proves amb els diferents sistemes de fitxers.

Tot això implica que cal instal·lar les eines d'extracció (Papiex i dependències), a més del propi sistema de cues, servidor de base de dades i servidor web. En el cas de Papiex, es van trobar alguns problemes de compatibilitat de Papiex i PAPI amb el processador de la màquina, que van solucionar-se parcialment gràcies a la col·laboració dels desenvolupadors actius d'aquestes eines. Per a la instal·lació de SLURM es va utilitzar la mateixa versió existent a MareNostrum, refent els fitxers de configuració per a adequar-los al sistema. Igualment, va ser necessària la configuració adient per a la base de dades i el servidor web per tal de posar-los en funcionament.

### 4.1.2 Clustertest

Per a proves de correctesa més serioses es troba clustertest. Com el nom indica, és un petit clúster format per tres nodes amb un entorn gairebé igual que el que es troba a MareNostrum. En concret, es tracta de *blades* JS-20, de dos processadors cadascun, que utilitzen la mateixa imatge de sistema operatiu que la del seu germà gran. Les diferències principals, a banda de la mida, són que en aquest clúster no hi ha xarxa Myrinet (tota la interconnexió es fa per la xarxa estàndard Gigabit Ethernet), ni GPFS (tot i que es munta el directori /home via NFS).

Dels tres nodes, un actua com a node mestre. En aquest node és on hi ha els sistemes de control de SLURM i el directori /home que s'exportarà als altres dos. Hi ha per tant un sistema de cues configurat i preparat per fer les proves necessàries comptant amb un entorn el més semblant possible a l'entorn real. Aquest clúster s'ha utilitzat per proves més serioses del *plugin* de SLURM i el sistema de recollida, ja que en aquest cas sí que es pot treballar amb nodes diferents i provar les

diferents opcions de generació de fitxers temporals als sistemes de fitxers. Com que físicament està dins de Nord, es pot arribar a fer servir un dels nodes de test com a servidor de recollida d'aquest altre clúster.

Anàlogament a l'esmentat a l'apartat anterior, s'han hagut d'instal·lar Papiex i les seves dependències, però aquest cop amb suport MPI. El servidor de recollida ha estat situat al node que fa controlador del sistema de cues (cabezatest), i la base de dades s'ha instal·lat a un dels servidors de Nord, on ja hi havia una instal·lació de MySQL. A més, s'ha hagut d'instalar OpenMPI per a realitzar proves amb una aplicació senzilla MPI, i retocar la configuració del gestor de recursos SLURM per instal·lar el *plugin*.

### 4.1.3 Nord

Aquest és el primer sistema en producció en el que s'ha treballat i s'han dut a terme proves. Nord és un clúster format per 81 *blades* JS-20, i com en el cas anterior, interconnectats també mitjançant una xarxa Gigabit Ethernet. En total, el clúster ofereix 162 processadors per a la recerca. En aquest clúster, amb usuaris reals, s'ha dut a terme la instal·lació de tot l'entorn necessari per a l'extracció de la informació i la integració en el sistema, per tal de fer proves en un entorn de producció real, pas previ a escalar fins a MareNostrum.

Per utilitzar Nord cal abans tenir cada part del programa provada en els entorns de proves anteriors per garantir que no s'interferirà amb els usuaris que hi treballen. De fet, la política definida és la de no realitzar cap tipus d'extracció sense fer-ho explícit al mateix treball que s'envia a les cues. Al tenir una mida mitjana i usuaris, es poden realitzar proves de capacitat del sistema a escala més controlada que en supercomputadors més grans i proves amb els programes habituals de recerca en aquestes instal·lacions.

Per problemes de compatibilitat, com es mencionava a la secció 3.3.4, cal utilitzar com a implementació MPI la que ofereix OpenMPI. El motiu és que Nord no compta amb una versió de la implementació MPICH compatible amb SLURM, pel que el funcionament del *plugin* construït per a carregar l'entorn no seria l'adequat. En aquest cas, també s'ha hagut d'instal·lar Papiex i dependències amb suport OpenMPI, a més d'instal·lar el *plugin*. A més, ha estat necessària la instal·lació de diverses aplicacions, efectuant la compilació utilitzant OpenMPI.

Per motius pràctics, s'ha mantingut la situació del servidor de recollida a cabezatest (ja que hi ha connexió directa des dels nodes de càlcul), i la base de dades al mateix servidor que en el cas anterior. En aquest últim aspecte ha estat necessària la instal·lació d'un nou servidor MySQL per a

suportar funcions definides per l'usuari i configurar-lo adequadament per a suportar els nivells de treball que el sistema generarà.

#### **4.1.4 MareNostrum**

Per últim, MareNostrum és el clúster més gran i l'objectiu que s'ha tingut en ment al idear aquest projecte. La descripció del supercomputador es troba a la secció 2.1. No hi ha data establerta per a la instal·lació de la infraestructura d'extracció, ja que cal primer sotmetre l'aplicació a un rigorós control d'estabilitat i funcionalitat en màquines més petites com Nord, pel que es necessita un temps.

De tota manera, a MareNostrum s'ha instal·lat la part d'obtenció de la informació de rendiment per comprovar-ne el funcionament. Així mateix, MareNostrum també s'ofereix per fer altres proves com les de rendiment dels sistemes de fitxers.

## **4.2 Experiments realitzats**

Tot seguit es descriuen els experiments i les proves realitzades per avaluar tota la infraestructura i les parts més destacades. També es troben altres experiments per comprovar l'eficiència de les solucions proposades, com és el cas de les proves relacionades amb la sobrecàrrega introduïda o el rendiment dels dos sistemes de generació de fitxers temporals a l'hora d'obtenir la informació.

### **4.2.1 Sobrecàrrega introduïda**

El primer punt a valorar per a posar una eina com aquesta en producció és l'impacte que té sobre el propi rendiment de les aplicacions que es mesuren. Donat que qualsevol mesura sobre un programa comporta una sobrecàrrega, és important mesurar-la per a veure si comporta una pèrdua significativa de temps d'execució en el programa.

En el cas particular del projecte, es mesura el temps d'execució de certes aplicacions tot obtenint-ne diferents conjunts de dades mitjançant l'eina Papiex. Els experiments s'han dut a terme a Nord, i utilitzant el sistema complet com a prova. Quant a la informació extreta, s'han utilitzat tres nivells diferents, amb diferent quantitat d'informació en cada cas, de menys a més, a part de l'execució sense obtenció d'informació (nivell 0). Aquests són els nivells definits per a aquestes proves:

**Level1**=PAPI\_TOT\_CYC, PAPI\_TOT\_INS

**Level2**=RUSAGE, MEMORY, PAPI\_TOT\_CYC, PAPI\_TOT\_INS, PAPI\_FP\_OPS

**Level3**=MULTIPLEX, RUSAGE, MEMORY, PAPI\_TOT\_INS, PAPI\_FP\_INS, PAPI\_LST\_INS,  
PAPI\_BR\_INS, PAPI\_INT\_INS, PAPI\_FMA\_INS, PAPI\_LD\_INS, PAPI\_SR\_INS, PAPI\_TOT\_CYC,  
PAPI\_L1\_DCM, PAPI\_L1\_ICM, PAPI\_TLB\_DM, PAPI\_TLB\_IM, PAPI\_L2\_DCM, PAPI\_L2\_ICM,  
PAPI\_STL\_ICY, PAPI\_BR\_MSP, PAPI\_FP\_OPS, PAPI\_L1\_DCA, PAPI\_L2\_ICA

Com a aplicacions s'han utilitzat el conjunt de *benchmarks* conegut com a NAS Parallel Benchmark[VW02] 2.4, que estan pensats per avaluar el rendiment de supercomputadors i que contenen càlculs que s'executen habitualment en altres aplicacions utilitzant MPI. El *benchmark* disposa de diverses mides de problema, havent escollit la classe B per ser la que més s'adequa a la quantitat de processadors que s'utilitzen i el temps desitjat de completat.

Les característiques de Nord fan que els temps d'execució d'un mateix treball puguin variar força, sobretot pel fet que no hi ha xarxa dedicada exclusivament per a les comunicacions. Aquest fet i l'arquitectura de xarxa del clúster fan que depenent dels nodes on s'executa el treball aquest tingui més o menys ample de banda disponible, a més de ser influenciat de forma imprevisible per la càrrega que provoquen altres treballs. Per això, aquestes proves s'han fet totes sobre el mateix conjunt de nodes, per intentar aïllar el soroll que aquest problema podria ocasionar sobre els resultats obtinguts.

A les taules 4.1, 4.2 i 4.3 es troben els resultats d'aquestes proves amb 4, 16 i 64 processadors respectivament. Cadascuna de les tres proves consisteix en executar els 7 programes que componen el *benchmark* quatre vegades, on a cada cop es fa servir un nivell diferent dels detallats anteriorment. La mida de la sobrecàrrega observada s'ha calculat amb la suma dels temps d'execució dels 7 programes relacionant-lo en cada cas amb el temps total de l'execució sense obtenció d'informació.

Com es pot veure, les diferències entre les execucions de cadascun dels components del *benchmark* amb o sense obtenció d'informació són mínimes. De fet, la mida de la sobrecàrrega observada oscil·la entre el -0.23% (16 CPUs, nivell 2) i el 1.50% (4 CPUs, nivell 3). Aquesta fluctuació indica que la sobrecàrrega és mínima, ja que queda emmascarada per la pròpia variació de la càrrega del sistema deguda a factors externs a l'aplicació mesurada. És per això que podem apreciar valors de sobrecàrrega negatius o molt propers a 0.

D'aquestes proves se'n desprèn el bon resultat que dona Papiex i reafirma en aquest cas la bona elecció de l'eina per a fer aquesta tasca dins del sistema. És cabdal que la sobrecàrrega introduïda sigui mínima, ja que aquest és un sistema pensat per a treballar sobre totes les aplicacions que estan

<b>4 CPUs</b>	<b>nivell 0</b>	<b>nivell 1</b>	<b>nivell 2</b>	<b>nivell 3</b>
<b>bt</b>	11 min 52.80 s	11 min 50.32 s	11 min 53.93 s	11 min 52.83 s
<b>cg</b>	2 min 11.32 s	2 min 11.61 s	2 min 12.24 s	2 min 13.41 s
<b>ep</b>	53.90 s	54.28 s	54.35 s	54.63 s
<b>is</b>	12.09 s	12.30 s	12.51 s	12.42 s
<b>lu</b>	6 min 31.34 s	6 min 33.28 s	6 min 31.34 s	6 min 36.96 s
<b>mg</b>	23.57 s	24.23 s	24.22 s	24.08 s
<b>sp</b>	9 min 59.39 s	10 min 13.72 s	10 min 10.34 s	10 min 18.90 s
<b>Temps total</b>	32 min 04.41 s	32 min 19.74 s	32 min 18.93 s	32 min 33.23 s
<b>Sobrecàrrega</b>	<b>0.00%</b>	<b>0.80%</b>	<b>0.75%</b>	<b>1.50%</b>

Taula 4.1: Resultats de l'execució de NAS classe B amb 4 processadors

<b>16 CPUs</b>	<b>nivell 0</b>	<b>nivell 1</b>	<b>nivell 2</b>	<b>nivell 3</b>
<b>bt</b>	3 min 51.12 s	3 min 52.47 s	3 min 53.56 s	3 min 53.46 s
<b>cg</b>	3 min 08.93 s	3 min 08.53 s	3 min 04.69 s	3 min 10.00 s
<b>ep</b>	14.82 s	16.97 s	15.78 s	15.45 s
<b>is</b>	8.85 s	9.31 s	9.57 s	9.29 s
<b>lu</b>	1 min 58.17 s	2 min 02.22 s	2 min 00.64 s	2 min 02.86 s
<b>mg</b>	18.21 s	17.84 s	18.63 s	18.23 s
<b>sp</b>	4 min 34.64 s	4 min 34.15 s	4 min 29.93 s	4 min 33.55 s
<b>Temps total</b>	14 min 14.74 s	14 min 21.49 s	14 min 12.80 s	14 min 22.84 s
<b>Sobrecàrrega</b>	<b>0.00%</b>	<b>0.79%</b>	<b>-0.23%</b>	<b>0.95%</b>

Taula 4.2: Resultats de l'execució de NAS classe B amb 16 processadors

<b>64 CPUs</b>	<b>nivell 0</b>	<b>nivell 1</b>	<b>nivell 2</b>	<b>nivell 3</b>
<b>bt</b>	3 min 22.24 s	3 min 29.61 s	3 min 23.49 s	3 min 25.02 s
<b>cg</b>	7 min 56.12 s	8 min 13.65 s	8 min 12.81 s	8 min 21.27 s
<b>ep</b>	5.35 s	6.19 s	6.29 s	6.24 s
<b>is</b>	1 min 00.78 s	1 min 00.24 s	53.31 s	1 min 03.62 s
<b>lu</b>	2 min 01.28 s	1 min 59.13 s	1 min 59.41 s	1 min 55.02 s
<b>mg</b>	58.79 s	52.60 s	48.88 s	48.68 s
<b>sp</b>	6 min 42.88 s	6 min 37.74 s	6 min 42.42 s	6 min 38.24 s
<b>Temps total</b>	22 min 7.44 s	22 min 19.16 s	22 min 6.61 s	22 min 18.09 s
<b>Sobrecàrrega</b>	<b>0.00%</b>	<b>0.88%</b>	<b>-0.06%</b>	<b>0.80%</b>

Taula 4.3: Resultats de l'execució de NAS classe B amb 64 processadors

en producció. Aquestes han de continuar rendint al mateix nivell com si no s'estiguessin mesurant, garantint un rendiment òptim per a tots els usuaris.

## 4.2.2 Sistemes de fitxers

Una de les modificacions introduïdes en el sistema ha estat la utilització dels discos locals (*scratch*) per a emmagatzemar els fitxers temporals generats per Papiex, explicada amb detall als apartats 3.3.2 i 3.3.3. En aquest punt es pretén comprovar el rendiment de les dues solucions, comparant-ne el rendiment quan s'utilitzen cert nombre de processadors.

Per a realitzar aquesta prova, s'ha desenvolupat un programa MPI senzill que conté la funcionalitat de realitzar la recollida des dels discos locals igual que la que incorpora Papiex. Així, és fàcil aïllar aquesta part i acotar-ne el temps. Cada procés dels que formaran l'execució tindrà prèviament copiats els fitxers necessaris als discos durs locals, els llegirà, els enviarà al procés mestre i aquest escriurà tots els fitxers al seu *scratch*.

També s'ha incorporat al programa un altre mode de funcionament que imita el comportament de Papiex quan treballa amb GPFS, on cada procés MPI escriu el seu fitxer sobre el mateix directori. Cal notar que en els dos casos s'han utilitzat fitxers reals de Papiex per a les proves, amb una mida de 1,6 KB. Aquestes proves s'han dut a terme a MareNostrum, on es disposa de la xarxa d'interconnexió Myrinet.

Per a validar els resultats i descartar valors erronis deguts a circumstàncies anormals del sistema, s'ha dut a terme cada prova un total de 50 vegades. Posteriorment, s'ha agafat com a temps de referència en cada cas la mitjana dels valors obtinguts.

Per un costat, s'analitza el comportament d'ambdós sistemes quan s'utilitzen pocs processadors (fins a 256). Els resultats es poden veure a la figura 4.1, on es mostra el temps que ha consumit cadascun dels sistemes en relació als processadors utilitzats. Queda palès que quan s'utilitzen pocs processadors, el mètode *scratch* és més ràpid que el de GPFS, fins i tot arribant a ser en el pitjor dels casos més del doble de ràpid. Aquesta diferència ve determinada sobretot perquè s'utilitza la xarxa Myrinet (el doble de ràpida que Gigabit Ethernet, i amb molta menys latència) per transmetre la informació. A més, l'escriptura sobre el disc local és més lleugera que sobre GPFS ja que aquest últim introdueix una sobrecàrrega inevitable per oferir el grau de paral·lelisme adient.

Per altra banda, cal veure què passa quan s'augmenta el nombre de processadors a cotes més altes. S'ha realitzat la comparativa fins a 2048 processadors, en la que s'observa de nou que el sistema desenvolupat per a l'ocasió segueix superant GPFS, tal i com es pot comprovar a la figura 4.2. Tot i

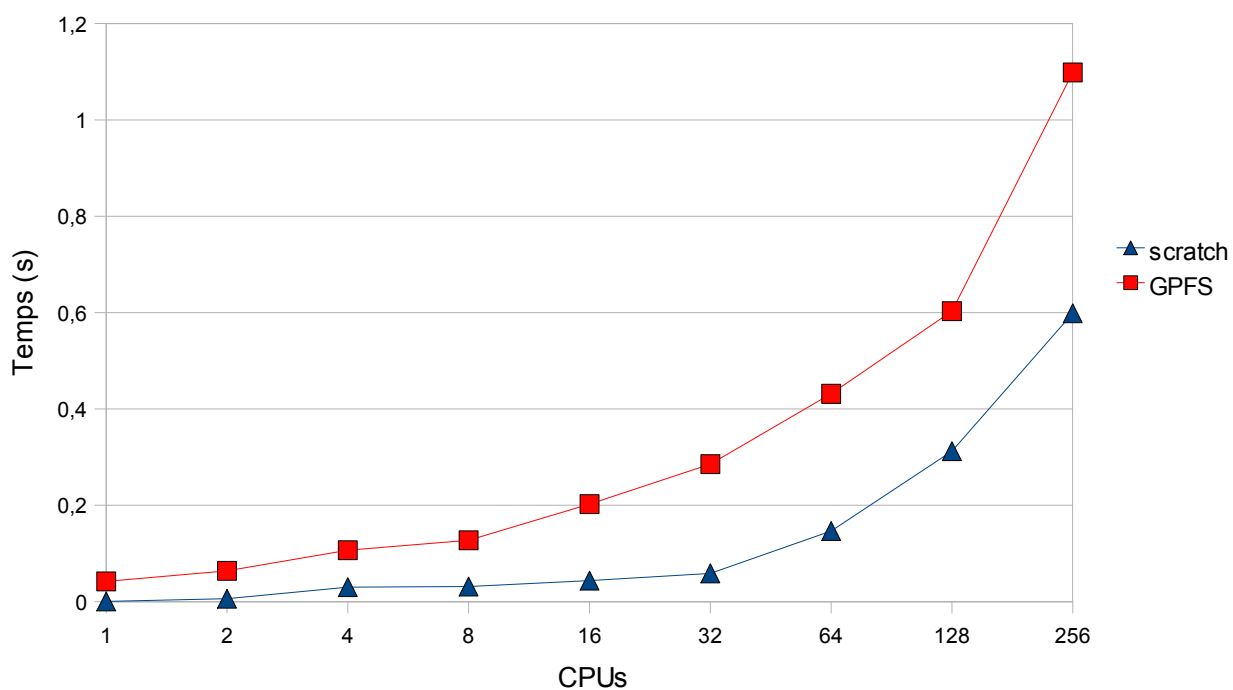


Figura 4.1: Comparativa de temps entre GPFS i *scratch* fins a 256 processadors

que l'escalabilitat en els dos casos es manté gairebé lineal, el mètode de recollida utilitzant *scratch* es demostra ser més ràpid en tots els casos analitzats fins a 2048 processadors. En aquest últim cas, per exemple, la diferència de temps ja és de 2.81 segons.

Com a complement a les figures mostrades, a la taula 4.4 es pot consultar el detall de les dades obtingudes.

Processadors	<i>scratch</i>	GPFS
1	0.56 ms	41.90 ms
2	6.26 ms	63.78 ms
4	29.88 ms	106.82 ms
8	31.41 ms	127.56 ms
16	43.66 ms	202.46 ms
32	58.71 ms	285.50 ms
64	146.67 ms	431.53 ms
128	312.56 ms	603.14 ms
256	598.54 ms	1.10 s
512	1.07 s	1.99 s
1024	2.08 s	3.69 s
2048	4.49 s	7.30 s

Taula 4.4: Comparativa de temps entre GPFS i *scratch*

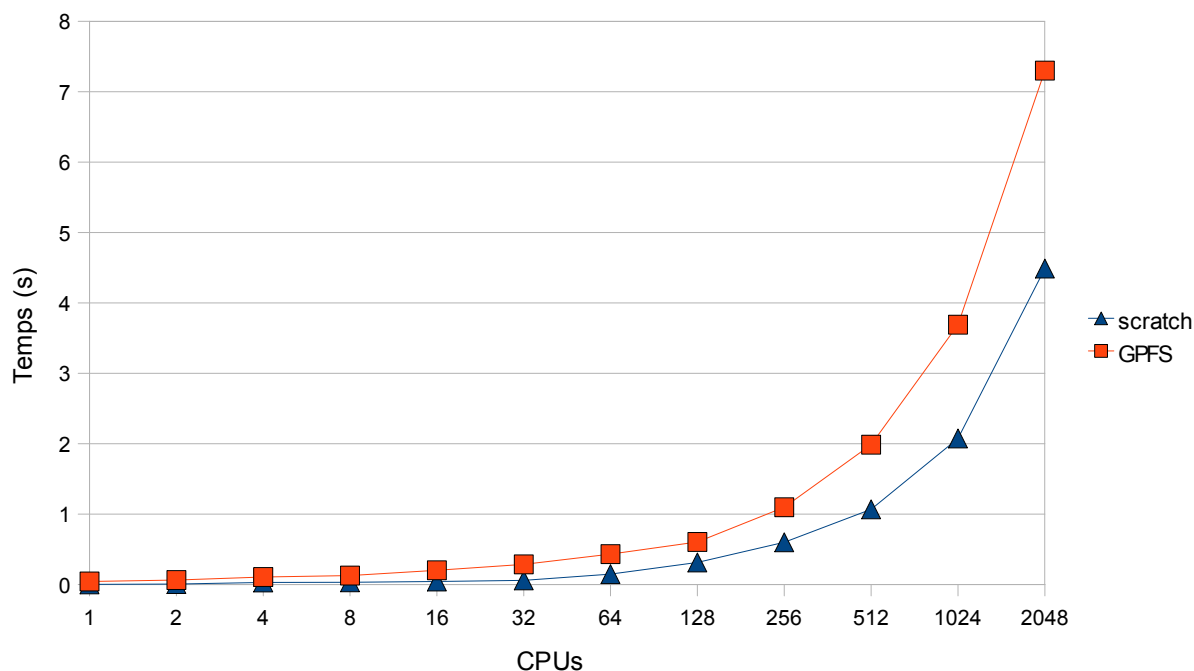


Figura 4.2: Comparativa de temps entre GPFS i *scratch* fins a 2048 processadors

### 4.2.3 Proves de capacitat

Tot i que a la secció 4.2.1 ja s'ha provat implícitament el funcionament correcte del sistema, cal veure quin és el comportament de tot el conjunt quan es sotmet a condicions més extremes. A partir d'aquest experiment es podrà assegurar un rendiment mínim, i ajudar a establir els paràmetres de configuració tant del *plugin* per a SLURM com en el servidor de recollida.

L'escenari d'aquestes proves torna a ser el sistema complet instal·lat a Nord. Tot i que la càrrega que es pot generar en aquest entorn no és comparable amb el que es podria generar a MareNostrum, és un bon punt de partida des del que justificar la seva viabilitat en clústers de grans dimensions. En concret, el que es vol mesurar és l'impacte en la càrrega mitjana de les màquines implicades en el funcionament de l'eina: el servidor de recollida i el servidor de base de dades. Aquests dos punts podrien arribar a ser el coll d'ampolla del sistema en condicions d'alta demanda des dels nodes de càlcul.

Les condicions més extremes en la que es pot trobar el sistema, com s'apuntava a l'apartat 3.5.2.1, es produeixen quan hi ha una gran quantitat de treballs amb duració molt curta. El motiu és senzill: quan s'acaba un treball, es produeix l'enviament de la informació i la seva recepció en el servidor de recollida. Cal recordar que la quantitat d'informació generada per cada procés és independent

de la seva durada, pel que amb treballs molt curts es generarà gran quantitat d'informació en poc temps.

La situació que es generarà és doncs l'enviament massiu de treballs el més curts possible, per tal d'obtenir un flux elevat de treballs que acaben i amb el nivell més alt d'extracció. Això hauria de suposar una càrrega molt més gran de l'habitual en el servidor de recollida, que haurà d'atendre moltes peticions en poc temps. A més, això significarà que hi haurà molta més informació emmagatzemada entre bolcat i bolcat a la base de dades, cosa que provocarà també que els bolcats siguin més voluminosos y més feixucs per al servidor on hi ha la base de dades.

El que es mesurarà, per a cadascun dels dos servidors, és el següent. Per una banda, la càrrega mitja del sistema en general, a l'hora que es fa un seguiment del procés particular implicat en la possible sobrecàrrega. En aquest cas, es mesura tant la utilització del processador com l'ús de la memòria. Per altra banda, també es mesurarà la utilització de la xarxa a cada servidor.

En total, s'envien 75 treballs a cues, de diverses mides (2, 4, 8, 16 i 32 processadors). S'envien de forma barrejada per a que a l'hora de la recollida hi hagi més heterogeneïtat. Des que el primer treball ha acabat fins que ho ha fet l'últim han passat 9 minuts i 5 segons, cosa que dóna una mitjana de 8.26 treballs/min, tot i que en alguns casos s'aconsegueix fer acabar treballs cada segon.

Pel que fa al servidor de recollida, a les figures 4.3 i 4.4 es troben la utilització de CPU (càrrega del sistema) i de la xarxa, respectivament. En el primer cas, es troben representades la càrrega del sistema, així com la utilització de CPU per part del procés responsable de la recollida. Aquesta informació s'ha obtingut mitjançant la comanda top, realitzant una enquesta d'aquests valors cada segon durant tot l'experiment. Com que les ràfegues d'utilització de CPU són molt curtes, és difícil reflectir amb exactitud la càrrega del procés concret, ja que pot passar que en el moment de consultar el consum, ja hagi acabat la seva ràfega. La utilització reportada seria de 0, mentre que en realitat si que hi hauria hagut un pic, que quedaria obviat.

S'observen pics d'utilització que coincideixen amb arribades més seguides de treballs, però la utilització màxima observada ha estat d'aproximadament del 31% d'un processador, però tenint en compte que aquest servidor disposa de 2 processadors, la utilització absoluta màxima que s'ha observat és la meitat, un 15.5%. Això ens indica que podríem arribar a carregar el sistema aproximadament 5 cops més sense que el rendiment es veiés afectat, ja que no s'arribaria al 80% d'utilització màxima. Per exemple, mantenint el mateix esquema de mides de treballs, es podria donar servei amb una freqüència d'arribada d'aproximadament 41 treballs/min (1,45 treballs per segon). Cal recordar, però, que l'arquitectura del sistema preveu que diversos servidors de recollida puguin donar servei, pel que l'escalabilitat d'aquesta solució estaria limitada pel nombre de servidors que estiguessin disponibles.

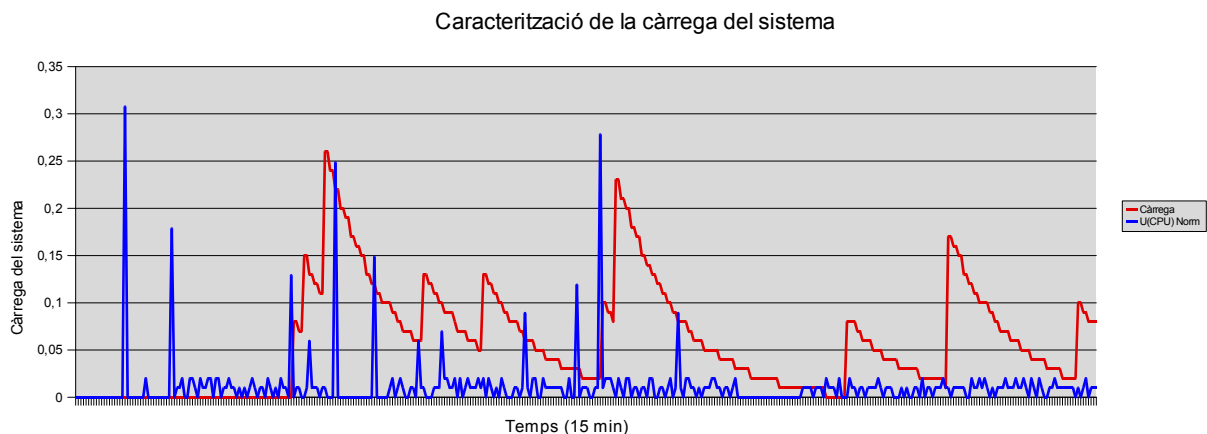


Figura 4.3: Utilització de CPU al servidor de recollida

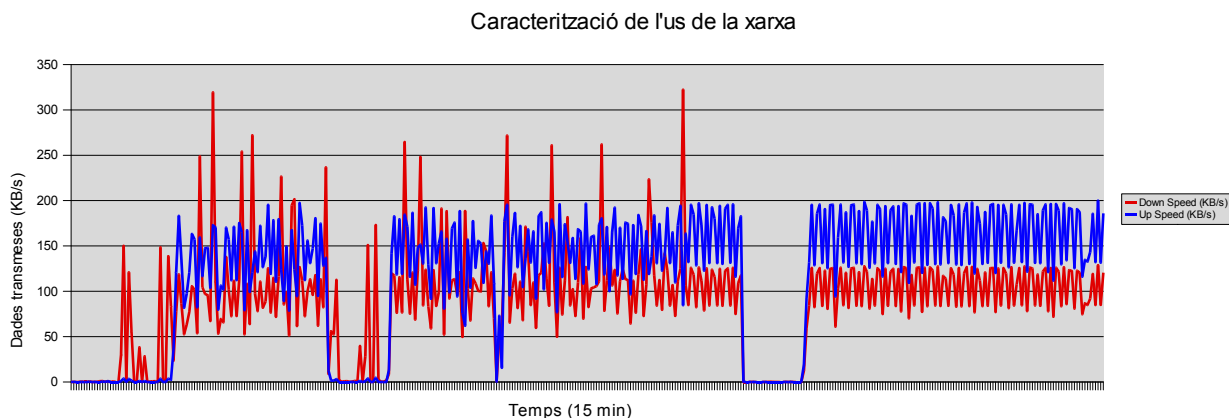


Figura 4.4: Utilització de la xarxa al servidor de recollida

De la mateixa manera, i per oferir un anàlisi més complert, s'ha calculat el temps de tractament mitjà d'un treball dins del servidor de recollida. Aquest temps inclou des que es comencen a rebre les dades des dels nodes de càlcul fins que s'ha tractat la informació i s'ha emmagatzemat temporalment a l'espera de ser bolcada a la base de dades. Bona part del temps es dedica a establir la connexió entre els dos servidors i realitzar l'enviament, mentre que una part més petita es destina al tractament de les dades.

Concretament, quan es tracta de mides de treballs de l'ordre de desenes de processadors o més, el temps per processador dedicat pel servidor de recollida són aproximadament 10 ms, dels quals només 1 ms es dedica al tractament de la informació rebuda. En treballs més petits, l'establiment de la connexió pesa més que la pròpia transferència i fa pujar el temps per processador fins als 35 ms.

Aquestes dades, però, no poden ser extrapolades a MareNostrum ja que l'arquitectura de xarxa és diferent. Tot i això, sabent que el temps de tractament és de 1 ms per processador, es pot observar fàcilment que, en un dels pitjors casos (un treball de 4000 processadors), el temps que es trigarà en rebre i tractar un treball serà de l'ordre de 4 segons. En aquest moment es pot produir una mica de contenció, ja que en aquests 4 segons és probable que acabi algun altre treball, que serà tractat més lentament.

Durant la realització d'aquesta prova, s'han realitzat tres bolcats cap a la base de dades. Cal tenir en compte que aquestes proves s'han fet amb el màxim nivell d'extracció, que implica haver d'emmagatzemar més de cent mètriques per a cada procés o *thread* del treball. Concretament, calen al voltant de 240 sentències d'inserció per a cada procés o *thread* que s'hagi d'emmagatzemar, i unes 970 sentències de consulta (necessàries per a obtenir les metadades que calguin a l'hora de realitzar cada inserció). A aquest nivell d'extracció, doncs, s'obté una taxa de 1,5 segons per emmagatzemar totes les dades d'un procés o *thread*.

Passant al servidor de base de dades, s'han realitzat les mesures de la mateixa forma que amb l'anterior. i es poden apreciar els resultats. S'ha de tenir en compte que aquest servidor, a més de la base de dades, gestiona el planificador (Maui) i el control de recursos (SLURM), pel que l'enviament massiu de treballs també afecta a aquests elements. A la figura 4.5 es pot comprovar aquest fet. Durant l'execució d'aquesta prova, s'arriba a posar el servidor en càrrega 4, que és la màxima que pot tolerar sense degradar el rendiment (El servidor compta amb 4 processadors). Es veu en canvi com el servei MySQL suposa una part mínima del total de càrrega que registra el servidor.

L'ús de la xarxa queda reflectit a la figura 4.6. Clarament, només s'observen pics durant els tres bolcats que tenen lloc des del servidor de recollida, i l'ús que se'n fa és gairebé residual (12.25 KB/s), ja que en els tres bolcats només s'han rebut 84 KB.

Aquest últim servidor és el candidat per esdevenir el coll d'ampolla del sistema, ja que a diferència dels servidors de recollida, no es pot replicar. Tanmateix, l'establiment d'un servidor de base de dades exclusivament dedicat faria difícil que el sistema es col·lapsés en aquest punt a la vista de les dades obtingudes. Queda demostrat, doncs, l'enorme avantatge de separar la generació de les dades del seu emmagatzemament mitjançant servidors de recollida replicables. Alhora que es dota de velocitat la recollida de la informació (perdent poc temps als nodes de càlcul per tractar la informació), s'evita en gran mesura que els pics de càrrega del sistema, quan acaben molts treballs seguits, afectin significativament el rendiment del supercomputador i de la base de dades.

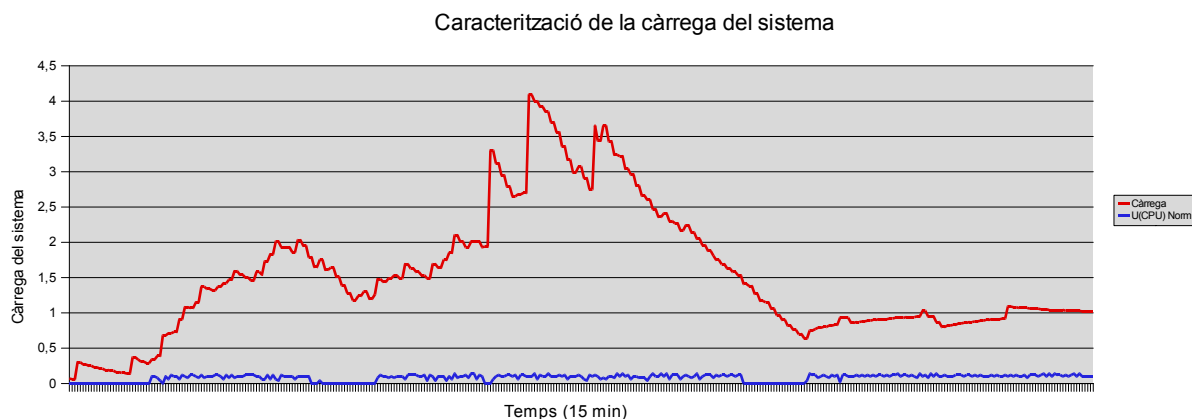


Figura 4.5: Utilització de CPU al servidor de base de dades

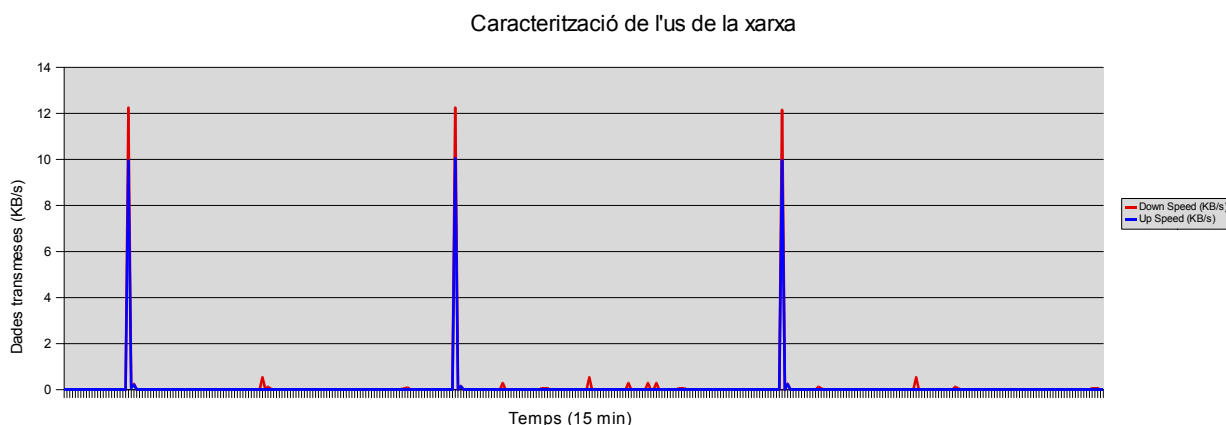


Figura 4.6: Utilització de la xarxa al servidor de base de dades

#### 4.2.4 Proves amb aplicacions habituals

Per últim, és convenient provar el sistema com un tot en condicions habituals. Això vol dir que diversos usuaris faran ús de diverses aplicacions instal·lades, i per tant se simularà una generació d'informació de rendiment provinent de diverses fonts a petita escala. La informació recollida durant aquestes execucions no només servirà per corroborar el bon funcionament del sistema de recollida i emmagatzemament de les dades, sinó també el seu anàlisi mitjançant l'aplicació web.

L'escenari d'aquestes proves tornarà a ser Nord, i com a actors principals prendran part membres de l'equip d'Operacions del BSC. S'ha escollit com a aplicacions tipus parts del benchmark NAS, utilitzat en anteriors proves, NAMD[NHG<sup>+</sup>96], un *software* de simulació de dinàmica mol·lecular i PEPC-B[Gib05], un simulador de la interacció entre làser i plasma.

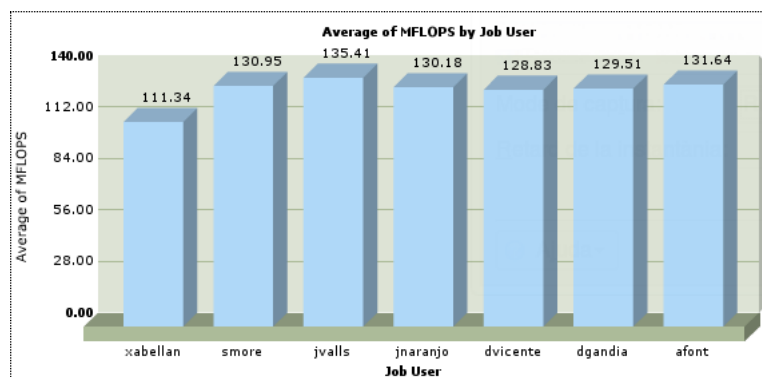


Figura 4.7: Gràfica que mostra els MFLOPS per usuari

Per facilitar les proves als usuaris participants, s'ha construït un *script* que simplifica la tasca de l'enviament i execució d'aquests treballs, de manera que només cal iniciar aquest script per a que l'usuari comenci a generar-los i executar-los. S'han adequat els paràmetres d'entrada dels programes i les mides dels problemes per a que siguin execucions relativament curtes (de menys d'una hora), suficient per al propòsit d'aquesta prova. També s'ha establert el nivell 3 d'extracció tal com està definit a l'apartat 4.2.1.

La realització d'aquesta prova ha estat un èxit, ja que els treballs s'han executat amb normalitat i en tots els casos s'ha recollit la informació que es desitjava. A continuació es mostren alguns exemples de l'anàlisi que es pot arribar a fer amb l'aplicació web. Cal remarcar que aquests exemples proposats són només alguns dels possibles usos de l'eina o les dades obtingues, de la mateixa manera que l'elecció de mètriques és molt més extensa que el que es mostra en els següents apartats.

#### 4.2.4.1 MFLOPS per usuari

A la figura 4.7 es pot apreciar que dels usuaris que han executat algun treball, n'hi ha que donen al voltant de 130 MFLOPS de mitjana, mentre que l'usuari xabellan està una mica per sota. Això és degut a que amb aquest usuari s'han dut a terme moltes altres proves que no són tant rendibles.

#### 4.2.4.2 MFLOPS per aplicació d'un usuari concret

En aquest cas s'observa quines aplicacions i quin rendiment en MFLOPS, de mitjana i de màxim, obté un usuari en concret de Nord. A la figura 4.8 es poden apreciar les diverses aplicacions que ha executat l'usuari dvicente. Es distingeixen les aplicacions que no realitzen càlcul, com *wc*, i aquelles que sí que en fan, de les quals destaca un dels programes del *benchmark* NAS, *ep*, amb 16

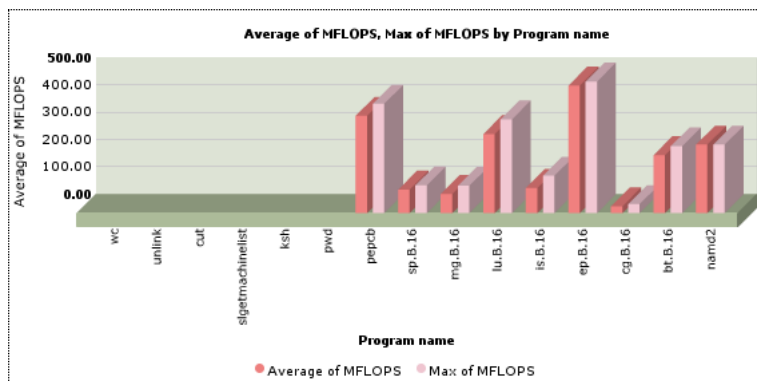


Figura 4.8: Gràfica que mostra els MFLOPS per cada programa utilitzat per un usuari

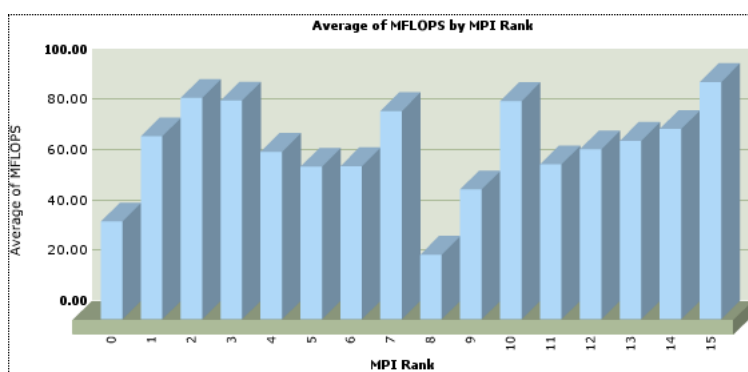


Figura 4.9: Gràfica que mostra els MFLOPS per cadascun dels processos MPI

processadors. En aquest gràfic es pot comparar el valor mitjà i el màxim obtingut de les execucions de cada programa

#### 4.2.4.3 Anàlisi d'una execució concreta

Un altre exemple d'ús d'aquesta eina seria veure el rendiment d'una aplicació en una execució concreta. En particular, es pren com a exemple una prova d'un dels components del *benchmark* NAS, mg, amb 16 processadors. Com s'aprecia a la figura 4.9, hi ha una diferència considerable entre el rendiment del càlcul que realitza cada procés MPI dins de l'aplicació. El que s'observa, ja que el temps utilitzat per cadascun d'ells és pràcticament igual, és que el càlcul no està uniformement repartit entre els processos. Això vol dir que n'hi han que fan moltes més operacions en coma flotant (el procés 15 fa prop de 95 MFLOPS) que d'altres (el procés 8 només és capaç de fer-ne poc més de 25).

Tot i això, es pot comprovar com no hi ha una relació directa amb fallades de memòria cau ni en el temps utilitzat en MPI. En aquest sentit, també s'observa un IPC similar en tots els processos de

l'execució MPI. Podria doncs tractar-se de les mateixes característiques del codi del programa, en el que alguns processos realitzen una part més important del càlcul. De tota manera, en un cas real, en aquest punt seria convenient dur a terme un estudi més a fons per entendre el comportament d'aquest programa, mitjançant eines més potents (i més específiques) com les vistes a la secció 2.4.

## Planificació i Anàlisi econòmic

L'objectiu d'aquest capítol és donar una idea del cost del treball realitzat durant el projecte. La primera part mostra la planificació pel projecte per a després comptar, per una banda, les hores dedicades a la realització del mateix i, per altra banda, el cost de l'equipament de la plataforma de mesures on s'han realitzat els experiments, donant finalment el total del cost econòmic que suposaria aquest projecte.

### 5.1 Planificació

En les següents línies es fa un repàs de la planificació temporal d'aquest projecte. A la figura 5.1 es troba el gràfic corresponent a aquesta planificació. El gràfic té una resolució setmanal, i va des de finals de gener del 2008 fins a finals de Setembre del mateix any. En ell es pot comprovar visualment l'evolució de les diferents parts que formen el projecte i la seva relació. A grans trets, s'ha dividit el projecte en cinc etapes principals i diferenciades: la concepció del projecte, el seu disseny, la implementació i adaptació, les proves i per la documentació del projecte, que inclou aquesta memòria.

Com a qualsevol projecte, cal començar amb una recerca d'informació sobre el tema, definir els objectius i acotar la feina a realitzar. Aquesta tasca és la que queda reflectida com a concepció del projecte. Inclou la recerca d'informació sobre l'estat de l'art, així com l'elecció de les eines a utilitzar. Un cop decidit prendre PerfMiner com a punt de partida, s'ha dut a terme un estudi més en profunditat del seu funcionament i s'han provat les eines d'extracció d'informació (Papiex) a l'entorn de producció. Cal tenir en compte que en aquestes proves es van detectar problemes de

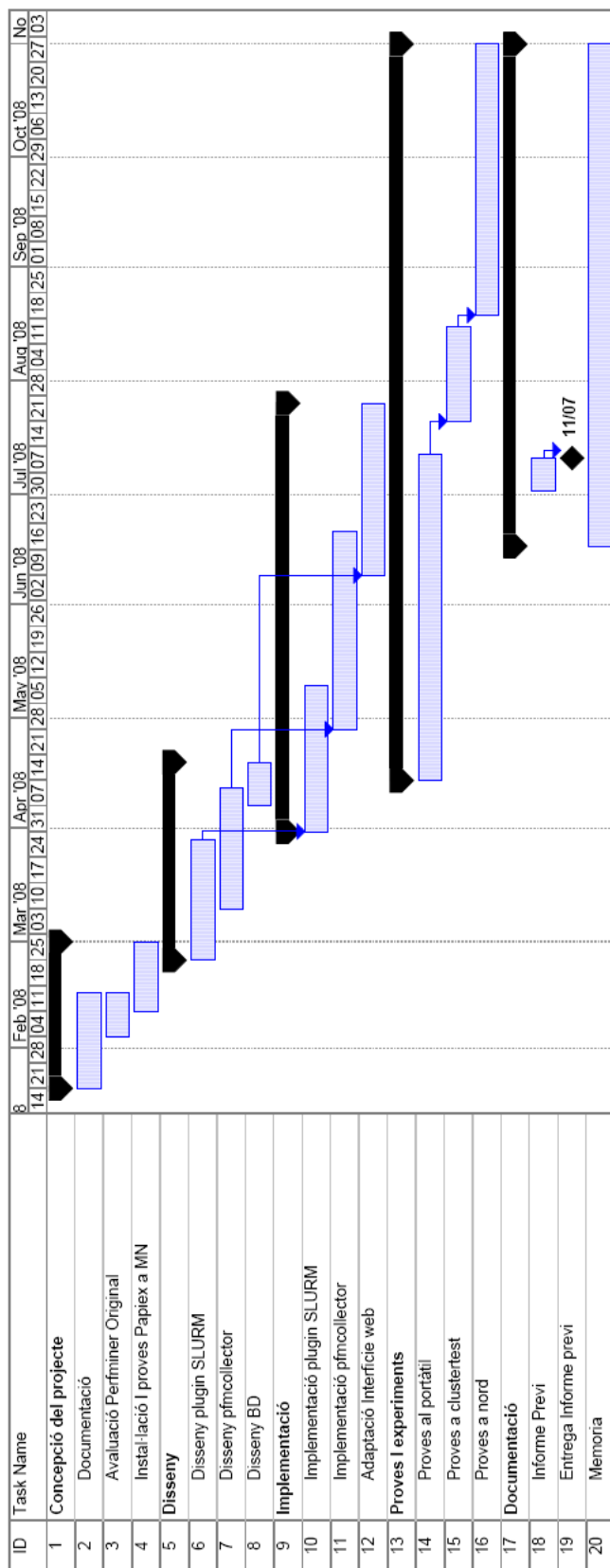


Figura 5.1: Planificació general del projecte

compatibilitat de l'eina a MareNostrum, pel que es va col·laborar amb els desenvolupadors per a tenir una versió funcional.

Posteriorment, es troben les fases de disseny i implementació, on s'ha dut a terme el desenvolupament de la infraestructura necessària i l'adaptació d'aquelles parts que ho requerien. Com es pot veure a la figura 5.1, aquestes dues fases s'han solapat parcialment. Això és degut a que el sistema està compost per diverses parts que s'han definit com a paquets de treball diferenciats. Tot i això la naturalesa del projecte fa que cada paquet tingui una relació estreta amb els altres, pel que s'han hagut de respectar les dependències lògiques entre ells i avançar la implementació d'un paquet al mateix temps que es realitza el disseny del següent.

Una de les fases més extenses ha estat la de les proves i experiments, ja que des del primer moment en que es té un prototipus de cada part s'ha començat a provar. Com s'apuntava a l'apartat 4.1, Les proves preliminars durant les primeres etapes del desenvolupament s'han dut a terme a l'equip portàtil, mentre que un cop es tenen versions prou madures de cada part, s'han pogut integrar en entorns més realistes com clustertest o nord.

Per últim, es contempla també l'etapa de documentació del projecte, que inclou tant aquesta memòria com l'informe previ que es va realitzar a principis de juliol. La memòria es va començar a escriure un cop acabat el gruix del desenvolupament del projecte, de manera que ja hi havia versions estables sobre les que treballar i els canvis des d'aquell moment havien de ser menors. Com es pot veure, s'ha compaginat la realització de la documentació amb la fase de proves i millores menors del sistema.

## 5.2 Anàlisi econòmic

En aquesta secció es comptabilitza el cost total de realització del projecte, comptant els costos relatius a la implementació de l'eina i els costos derivats de la realització dels experiments.

### 5.2.1 Cost de la implementació

El cost d'implementació incorpora les hores d'anàlisi, programació i desplegament destinades al desenvolupament de les diverses parts del projecte. Això inclou les modificacions de Papiex, el *plugin* per a SLURM, el sistema de recollida, l'adaptació de la base de dades i de l'aplicació web per a visualitzar els resultats. Les hores invertides es divideixen en tres perfils:

- **Analista:** Persona encarregada de dissenyar i planificar les tasques de programació.

- **Programador:** Persona que realitza totes les tasques de programació i les que es deriven d'aquestes per a posar en marxa l'aplicació.
- **Tècnic de sistemes:** Persona que realitza el desplegament de la infraestructura i en realitza l'administració.

La taula 5.1 mostra el cost d'implementació del sistema complet.

Perfil	Hores	Preu/hora	Total
Analista	125	60 €	7500 €
Programador	345	40 €	13800 €
Tècnic de Sistemes	90	50 €	4500 €
<b>Total</b>			25800 €

Taula 5.1: Cost d'implementació

## 5.2.2 Cost de l'equipament

El següent apartat correspon al cost de l'equipament utilitzat per realitzar el projecte. Com s'ha vist a la secció 4.1, els equips utilitzats han estat el portàtil, un petit clúster de proves, un clúster mitjà (Nord) i un clúster gran (MareNostrum). L'equip portàtil és també l'eina de treball per a realitzar altres tasques a part del projecte, i es pren com a període d'amortització d'aquest 3 anys, on l'ús aproximat d'aquest han estat 6 mesos. Per tant, l'amortització d'aquest equipament és del 33.3%.

Donat que els recursos de supercomputació que han intervingut en el projecte no s'han utilitzat ni molt menys exclusivament per a aquest treball, es comptabilitza el seu ús segons les hores de càlcul utilitzades tal com si es tractés d'un usuari extern. No tindria sentit incloure en els costos del projecte el cost d'adquisició i manteniment dels grans supercomputadors de que disposa el centre. S'assumeix també que els servidors que donen suport a la infraestructura (servidor de recollida, base de dades i web) formen part del supercomputador en sí, ja que són equips que ja donen servei habitualment.

La taula 5.2 mostra els costos de l'equipament per al desenvolupament del projecte.

Concepte	Unitats	Amortització	Cost	Total
Equip portàtil	1	0.33	1200 €	400 €
Hores de supercomputador	500	1	0.50 €	250 €
<b>Total</b>				650 €

Taula 5.2: Cost de l'equipament utilitzat

### 5.2.3 Cost total

Finalment, el cost total del projecte es reflecteix a la taula 5.3 amb la suma dels costos d'implementació i els costos derivats de l'equipament.

<b>Concepte</b>	<b>Cost</b>
Cost implementació	25800 €
Cost equipament	650 €
<b>Total</b>	<b>26450 €</b>

Taula 5.3: Cost total del projecte



## Conclusions i Treball Futur

En aquest capítol final del projecte es fa un breu resum del treball realitzat i s'exposen les conclusions que se'n poden extreure. La primera part del capítol està enfocada a resumir les tasques realitzades i la següent part a descriure les futures aplicacions i el treball que pot derivar d'aquest projecte.

### 6.1 Sumari

L'objectiu d'aquest projecte era aconseguir un sistema capaç d'obtenir informació sobre el rendiment de les aplicacions que s'executen en un supercomputador com MareNostrum de manera transparent i totalment integrada amb el sistema i configuració originals. A més, havia d'oferir un mètode per a emmagatzemar aquesta informació per a poder analitzar posteriorment el rendiment que se'n treu del supercomputador a diferents nivells, tenint en compte tant el que s'executa, com qui ho executa i en quines circumstàncies.

Es pot afirmar doncs que s'ha assolit la meta marcada a l'inici del projecte. El resultat ha estat l'adaptació i millora de PerfMiner, una eina existent però amb possibilitats limitades d'implantació en entorns com els que hi ha al centre. Gràcies a això, s'han pogut reaprofitar els punts forts d'elements ja existents, com la tasca d'obtenció de la informació de rendiment a més baix nivell o la base de dades per emmagatzemar-la, tot afegint en cada cas les millores oportunes.

Alhora, aquelles parts que no podien funcionar adequadament donades les característiques de la configuració dels supercomputadors dels que es disposen s'han adaptat o refet totalment. Aquest

és el cas de la integració amb el sistema, aprofitant les possibilitats ofertes pel gestor de recursos SLURM o també el de la recollida de la informació. En aquest punt s'ha replantejat el disseny de la recollida per a fer-la més escalable i operativa per a supercomputadors de gran mida. També la part d'anàlisi ha rebut algunes millores respecte la versió original, ja que s'han incorporat algunes funcionalitats i opcions que resulten interessants per a dur a terme un anàlisi de les dades obtingudes més acurat.

Com a prova del seu correcte funcionament, s'ha desplegat el sistema en un supercomputador molt semblant a MareNostrum, com és Nord, però a més petita escala. Les proves realitzades han demostrat la seva bona integració amb el sistema i la seva configuració normal sense introduir una sobrecàrrega significativa en cap de les parts en les que s'ha realitzat alguna actuació. Per tant, els objectius de l'eficiència i la mínima interferència amb el funcionament habitual de la infraestructura utilitzada també s'han aconseguit.

D'altra banda, ha quedat pendent la instal·lació d'aquesta eina a l'entorn per la qual ha estat dissenyada: MareNostrum. Tot i que no s'ha provat en un supercomputador tan gran, l'experiència adquirida amb la instal·lació a Nord i l'evolució del seu funcionament fan pensar que no hi hauria d'haver cap problema per a instal·lar el sistema a MareNostrum. Cal destacar l'oportunitat que oferirà aquesta eina un cop instal·lada, per descobrir l'ús real que es fa dels recursos de supercomputació, detectar possibles problemes i ajudar als usuaris en la seva tasca diària de recerca.

Deixant de banda l'aspecte més pràctic, és important fer notar el caràcter heterogeni d'aquest projecte. S'han abordat diverses àrees de la informàtica, des del nivell de l'obtenció de la informació de rendiment fins a la programació d'una aplicació web per al seu anàlisi. S'han aplicat conceptes de sistemes operatius, enginyeria del *software*, programació en diversos llenguatges, etc. Tot plegat fa que aquest projecte hagi estat molt enriquidor.

Per últim, no es pot tancar aquest sumari sense recalcar la filosofia oberta que ha guiat aquest projecte. Des del principi, s'han integrat elements d'altres persones per obtenir un sistema nou i millor. Fruit d'això, durant aquest procés s'ha establert una bona relació amb aquestes persones, que a més d'aportar parts fonamentals del sistema, s'han mostrat disposades a incorporar les millores que aquest projecte ha introduït a les respectives versions oficials.

## 6.2 Treball futur

El sistema resultant d'aquest projecte no és més que una primera versió, que tot i ser prou funcional i satisfer les necessitats proposades, es pot millorar o ampliar en diversos aspectes. Bona part de les

propostes que s'exposen tot seguit tenen com a principals motius objectius que no s'han considerat prioritari per aquest projecte, però que podrien ser interessants, com la portabilitat. Algunes d'aquestes propostes ja han estat exposades al llarg de la descripció del projecte, però pot resultar útil recollir-les totes, ja que han de marcar el camí a seguir en cas que es volgués seguir treballant en aquesta eina.

El primer dels punts que s'ha quedat al tinter és la integració de mpiP dins d'aquest projecte, conjuntament amb l'eina d'obtenció d'informació Papiex. Amb aquesta nova eina, es podria obtenir dades de quin és l'ús de MPI que realitzen les aplicacions analitzades. Per aconseguir aquest objectiu caldria adaptar d'alguna manera el *plugin* SLURM i el sistema de recollida per tal que fossin capaços d'entendre i identificar correctament les dades generades per mpiP.

En segon lloc, es podria dotar de més robustesa al procés de recollida. Principalment, es podria incorporar elements de seguretat per autenticar la connexió entre els nodes i els servidors de recollida. Probablement xifrar la comunicació sencera no és una opció desitjable, per la sobrecàrrega de càlcul que suposa, però sí que pot interessar autenticar la connexió per després enviar les dades sense xifrar. Per fer això es podria mirar d'utilitzar el mateix sistema d'autenticació de SLURM (*munge*). En la mateixa línia, es podria millorar el protocol de comunicació utilitzat en el procés de recollida. Mentre que en la versió present s'utilitza una comunicació a nivell de *sockets*, podria resultar interessant utilitzar un protocol de RPC (*Remote Procedure Call*), proporcionant una major estandardització del codi.

Quant als servidors de recollida, després d'avaluar el rendiment de la versió actual, una possibilitat seria portar el codi a un llenguatge orientat a objectes com C++, que podria oferir una implementació d'una eficiència semblant, i alhora es dotaria al programa de major modularitat i mantenibilitat del codi. De la mateixa manera, una bona ampliació seria també utilitzar un mètode més abstracte d'accés a la base de dades, per tal que no depengui del sistema gestor que s'utilitza en cada cas.

També la part de visualització i anàlisi presenta oportunitats per a l'ampliació i millora. Per una banda la qüestió estètica es podria millorar amb la utilització de fulls d'estil més sofisticats. Per altra, es pot guanyar en flexibilitat afegint noves opcions per a l'usuari de l'eina. La necessitat d'incorporar aquestes noves funcionalitats pot anar sorgint a mida que es vagi fent ús d'aquesta eina i s'observi la falta d'algun element o la possible millora d'un cert procediment d'anàlisi. De la mateixa manera, es pot pensar en noves aplicacions que treballin sobre la base de dades amb la informació. Un exemple seria un generador d'informes de rendiment per a certs grups o usuaris, per informar de l'ús que estan fent dels recursos d'una manera regular.

Finalment, un dels objectius generals a aconseguir en aquest treball futur seria la integració cada cop més intensa del desenvolupament de les diferents parts d'aquest projecte. S'hauria de mantenir

la comunicació i la bona sintonia amb els desenvolupadors de les eines originals per aconseguir entre tots un sistema molt més fiable, eficient i útil per a tothom.

## Descripció de mètriques derivades

Algunes de les mètriques disponibles a través de PerfMiner i Papiex són derivades. Es creen a partir d'altres mètriques, i aporten una nova informació o visió d'un cert aspecte del programa. Per tal de facilitar la comprensió d'aquestes mètriques, s'han recollit en aquest annex els orígens de cadascuna.

- **Branch Instructions %:** Tant per 1 d'instruccions de salt.

$$\frac{CPU\_BRANCH}{CPU\_INSEXC}$$

- **Branch Misprediction %:** Tant per 1 de salts mal predits.

$$\frac{CPU\_MISPRED}{CPU\_BRANCH}$$

- **Computational Intensity:** Flops per load/store.

$$\frac{CPU\_FPARITH + CPU\_FPMADD}{CPU\_LOAD + CPU\_STORE}$$

- **CPU Utilization:** Cicles d'execució / Cicles transcorreguts.

$$\frac{VIRTUAL\_CYCLES}{REAL\_CYCLES}$$

- **D-TLB Hit %:** Tant per 1 d'encert a la TLB de dades.

$$1 - \frac{CPU\_DTLBMIS}{CPU\_LOAD + CPU\_STORE}$$

- **Flops per D-cache Miss:** Operacions en punt flotant per cada fallada a la cau de dades.

$$\frac{CPU\_FPARITH + CPU\_FPMADD}{CPU\_DCMISS}$$

- **FMA Instructions %:** Tant per 1 d'intruccions en punt flotant multiplicació-suma.

$$\frac{CPU\_FPMADD}{CPU\_INSEXC}$$

- **FP Arith. Ins %:** Tant per 1 d'instruccions aritmètiques en punt flotant.

$$\frac{CPU\_FPARITH}{CPU\_INSEXC}$$

- **FP ins. per D-cache Miss:** Instruccions en punt flotant per cada fallada a la cau de dades.

$$\frac{CPU\_FLOAT}{CPU\_DCMISS}$$

- **FP Instructions %:** Tant per 1 d'instruccions ens punt flotant (incloent ld/st).

$$\frac{CPU\_FLOAT}{CPU\_INSEXC}$$

- **I-TLB Hit %:** Tant per 1 d'encerts a la TLB d'instruccions.

$$1 - \frac{CPU\_ITLBMIS}{CPU\_LOAD + CPU\_STORE}$$

- **Integer Instructions %:** Tant per 1 d'instruccions amb enters (incloent ld/st).

$$1 - \frac{CPU\_INTEGER}{CPU\_INSEXC}$$

- **I/O Cycles %:** Tant per 1 de cicles gastats en entrada/sortida.

$$\frac{IO\_CYC}{REAL\_CYCLES}$$

- **IPC:** Instruccions per cicle.

$$\frac{CPU\_INSEEXEC}{CPU\_CYCLES}$$

- **L1 I-cache Hit %:** Tant per 1 de fallades a la cau L1 d'instruccions

$$1 - \frac{CPU\_ICMISS}{CPU\_LOAD + CPU\_STORE}$$

- **L1 D-cache Hit %:** Tant per 1 de fallades a la cau L1 de dades

$$1 - \frac{CPU\_IDMISS}{CPU\_LOAD + CPU\_STORE}$$

- **L2 Miss Ratio:** Fallades a la cau L2 per a cada 1000 instruccions.

$$\frac{CPU\_L2MISS}{CPU\_INSEEXEC} \cdot 1000$$

- **Loads/Stores Ratio:** Proporció de loads i stores.

$$\frac{CPU\_LOAD}{CPU\_STORE}$$

- **Memory Instructions %:** Tant per 1 d'instruccions d'accés a memòria.

$$\frac{CPU\_LOAD + CPU\_STORE}{CPU\_INSEEXEC}$$

- **MFLOPS:** Milions d'operacions en punt flotant per segon.

$$\frac{(CPU\_FPARITH + CPU\_FPMADD) \cdot f(MHz)}{VIRTUAL\_CYC}$$

- **MFLIPS:** Milions d'instruccions en punt flotant per segon.

$$\frac{CPU\_FLOAT \cdot f(MHz)}{VIRTUAL\_CYC}$$

- **MPI Cycles %:** Tant per 1 de cicles gastats en MPI.

$$\frac{MPI\_CYC}{REAL\_CYCLES}$$



---

# Glossari de Termes

**AIX** Advanced Interactive eXecutive. Sistema operatiu propietari de IBM basat en UNIX.

**API** Application Program Interface, interfície de programació d'aplicacions.

**BSC-CNS** Barcelona Supercomputing Center - Centro Nacional de Supercomputación.

**Clúster** Conjunt d'ordinadors que es comporten com si fossin un de sol.

**CPU** Central Processing Unit o Unitat Central de Processament. CPU és sinònim de processador.

**DMA** Direct Memory Access. Accés directe a memòria per part del maquinari per a no haver d'utilitzar la CPU.

**Flop** Operació en coma flotant per segon.

**GPL** GNU General Public License, principal llicència de programes de codi obert.

**Linpack** Benchmark que es pren com a referència per mesurar el rendiment d'un supercomputador.

**Log** Dietari on es registren les diferents accions d'un programa.

**Memory leaks** Problema amb l'ús de la memòria per part d'un programa, que no allibera la que ja no utilitza.

**MPI** Message Passing Interface. Interfície de pas de missatges per a programes paral·lels.

**OpenMP** Paradigma de programació paral·lela amb memòria compartida (threads).

**POSIX** Portable Operating System Interface, és una família d'estàndards definits per IEEE per a definir una API comuna per a sistemes UNIX.

**Profile** Perfil de comportament d'una aplicació.

Shell Intèrpret de comandaments.

Speed-up Guany que té una aplicació quan s'augmenten els recursos assignats per a la seva execució.

Thread Fil d'execució dins d'un procés.

---

# Índex de figures

2.1	Bucle seqüencial senzill . . . . .	12
2.2	Bucle senzill paral·lelitzat amb OpenMP . . . . .	12
2.3	Programa mpi que es comunica seguint la forma d'un anell . . . . .	14
2.4	Exemple de treball paral·lel a MareNostrum . . . . .	19
2.5	Fragment de sys/resource.h on es descriu l'estructura rusage . . . . .	24
2.6	Interposició d'una funció per instrumentar-la . . . . .	25
3.1	Estructura general del sistema a desenvolupar . . . . .	39
3.2	Exemple d'arxiu generat per Papiex . . . . .	43
3.3	Diagrama de la API SPANK . . . . .	52
3.4	Diagrama de seqüència dels <i>plugins</i> SPANK . . . . .	53
3.5	Diagrama de classes del <i>plugin</i> spank_perfminer . . . . .	55
3.6	Diagrama de seqüència de la inicialització al <i>plugin</i> spank_perfminer . . . . .	57
3.7	Diagrama de seqüència de la finalització al <i>plugin</i> spank_perfminer . . . . .	58
3.8	Distribució dels <i>slots</i> temporals entre transferències . . . . .	67
3.9	Distribució dels <i>slots</i> temporals entre transferències per mida de transferència . . . . .	68
3.10	Diagrama de classes de pfmcollector . . . . .	70
3.11	Diagrama de seqüència del procés de <i>flush</i> . . . . .	74
3.12	Esquema de la base de dades . . . . .	77
3.13	Diagrama de classes de la interfície web de PerfMiner . . . . .	82

---

3.14	Vista principal de l'aplicació web per a la visualització . . . . .	85
4.1	Comparativa de temps entre GPFS i <i>scratch</i> fins a 256 processadors . . . . .	94
4.2	Comparativa de temps entre GPFS i <i>scratch</i> fins a 2048 processadors . . . . .	95
4.3	Utilització de CPU al servidor de recollida . . . . .	97
4.4	Utilització de la xarxa al servidor de recollida . . . . .	97
4.5	Utilització de CPU al servidor de base de dades . . . . .	99
4.6	Utilització de la xarxa al servidor de base de dades . . . . .	99
4.7	Gràfica que mostra els MFLOPS per usuari . . . . .	100
4.8	Gràfica que mostra els MFLOPS per cada programa utilitzat per un usuari . . . . .	101
4.9	Gràfica que mostra els MFLOPS per cadascun dels processos MPI . . . . .	101
5.1	Planificació general del projecte . . . . .	104

---

# Índex de taules

3.1	Temps teòrics de transferència i mides aproximades dels fitxers temporals . . . . .	66
3.2	Distribució dels <i>slots</i> temporals i el percentatge del total de les transferències . . . . .	67
4.1	Resultats de l'execució de NAS classe B amb 4 processadors . . . . .	92
4.2	Resultats de l'execució de NAS classe B amb 16 processadors . . . . .	92
4.3	Resultats de l'execució de NAS classe B amb 64 processadors . . . . .	92
4.4	Comparativa de temps entre GPFS i <i>scratch</i> . . . . .	94
5.1	Cost d'implementació . . . . .	106
5.2	Cost de l'equipament utilitzat . . . . .	106
5.3	Cost total del projecte . . . . .	107



---

## Bibliografia

- [Amd62] G.M. Amdahl. New concepts in computing system design. *Proceedings of the IRE*, 50(5):1073–1077, May 1962.
- [Art06] E. Artiaga. Ggcollector. <http://www.bsc.es>, 2006.
- [Art07] E. Artiaga. Monitoring infrastructure for superclusters: Experiences at marenostrom. <http://www.bsc.es>, 2007.
- [BDG<sup>+</sup>00] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proc. ACM/IEEE 2000 Conference Supercomputing*, pages 42–42, 04–10 Nov. 2000. <http://icl.cs.utk.edu/papi/>.
- [BH00] B. Buck and J.K. Hollingsworth. An API for Runtime Code Patching. *International Journal of High Performance Computing Applications*, 14(4):317, 2000. <http://www.dyninst.org/>.
- [DeR01] L. DeRose. The Hardware Performance Monitor Toolkit. *Proceedings of Euro-Par*, pages 122–131, 2001.
- [DM98] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, Jan.–March 1998. <http://www.openmp.org/>.
- [DZ83] J.D. Day and H. Zimmermann. The osi reference model. 71(12):1334–1340, Dec. 1983.
- [F S02] R Haskin F Schmuck. Gpfs: A shared-disk file system for large computing clusters. *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, 19, 2002. <http://www.ibm.com/systems/clusters/software/gpfs.html>.

- 
- [Fus] Fusioncharts free. [www.fusioncharts.com/free/](http://www.fusioncharts.com/free/).
- [Gan] The ganglia scalable distributed monitoring system. <http://ganglia.sourceforge.net>.
- [Gib05] P. Gibbon. PEPC: A Multi-Purpose Parallel Tree-Code, 2005. <http://www.fz-juelich.de/jsc/pepc/>.
- [GLDS96] W. Gropp, Lusk E., N. Doss, and A. Skjellum. *A High Performance Portable Implementation of the MPI Message Passing Interface Standard*, 1996. <http://www-unix.mcs.anl.gov/mpi/mpich1/docs.html>.
- [GWS05] Richard L. Graham, Timothy S. Woodall, and Jeffrey M. Squyres. Open MPI: A flexible high performance MPI. In *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Poland, September 2005. <http://www.open-mpi.org/>.
- [Kuf05] R. Kufirin. PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. *Presented at The 6th International Conference on Linux Clusters: The HPC Revolution*, 151:05, 2005. <http://perfsuite.ncsa.uiuc.edu/>.
- [MAD<sup>+</sup>05] P.J. Mucci, D. Ahlin, J. Danielsson, P. Ekman, and L. Malinowski. PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data. *Proceedings of 2005 European Conference on Parallel Computers (Euro-Par)*, pages 124–133, 2005. <http://perfminer.pdc.kth.se/>.
- [Mes95] The Message Passing Interface Forum (MPIF). *MPI: A Message-Passing Interface Standard*, 1995. <http://www.mpi-forum.org/docs/>.
- [Mic07] Sun Microsystems. Lustre file system: High-performance storage architecture and scalable cluster file system. 2007. <http://www.sun.com/>.
- [MNA] Marenostrom architecture. <http://www.bsc.es>.
- [Muc05] PJ Mucci. papiex: Transparently Measure Hardware Performance Events of an Application with PAPI, 2005. <http://icl.cs.utk.edu/mucci/papiex/>.
- [Myr] Myrinet overview. <http://www.myri.com/myrinet/overview/>.
- [MyS06] AB MySQL. MySQL 5.0 Reference Manual, 2006. <http://dev.mysql.com/doc/refman/5.0/en/>.
-

- [NAW<sup>+</sup>96] W.E. Nagel, A. Arnold, M. Weber, H.C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and Analysis of MPI Resources. *Supercomputer*, 12(1):69–80, 1996. <http://www.vampir.eu/>.
- [NHG<sup>+</sup>96] M.T. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L.V. Kale, R.D. Skeel, and K. Schulten. NAMD: a Parallel, Object-Oriented Molecular Dynamics Program. *International Journal of High Performance Computing Applications*, 10(4):251, 1996.
- [Per] J. Persson. JpGraph-OO Graph Library for PHP. <http://www.aditus.nu/jpgraph/>.
- [PLC<sup>+</sup>95] V. PILLET, J. LABARTA, T. CORTES, S. GIRONA, and D.A. de Computadors. PARAVÉR: A Tool to Visualize and Analyze Parallel Code. *Transputer and OCCAM Developments: WoTUG-18: Proceedings of the 187th World Occam and Transputer User Group Technical Meeting, 9th-13th April 1995, Manchester, UK*, 1995. <http://www.bsc.es/>.
- [Res] Cluster Resources. Moab workload manager administrator's guide. <http://www.clusterresources.com>.
- [SM06] S. Shende and AD Malony. TAU: The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006. <http://www.cs.uoregon.edu/research/tau/home.php>.
- [SN] J. Seward and N. Nethercote. Valgrind, an open-source memory debugger for x86-GNU. *Linux*. available at URL <http://developer.kde.org/sewardj>. <http://valgrind.org/>.
- [top] Top500 supercomputing sites. <http://www.top500.org>.
- [Tor02] Linus Torvalds. Linux Kernel, 2002. <http://www.kernel.org>.
- [VC05] J. Vetter and C. Chabreau. mpiP: Lightweight, Scalable MPI Profiling. 2005. <http://mpip.sourceforge.net/>.
- [VW02] R.F. Van der Wijngaart and P. Wong. NAS Parallel Benchmarks Version 2.4. *NASA Ames Research Center: NAS Technical Report NAS-02-007*, 2002.
- [wik] Wikipedia. <http://www.wikipedia.org/>.
- [YJG03] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn,

editors, *JSSPP*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003. <https://computing.llnl.gov/linux/slurm/>.